## Contents

*What's New in SAS 9.4 Data Quality Server* .................................................. v

**Chapter 1 • Overview of SAS 9.4 Data Quality Server** ................................. 1  
  Capabilities and Requirements ......................................................... 1  
  Integration ......................................................................................... 2  
  Resources ......................................................................................... 3

**Chapter 2 • Getting Started with SAS Data Quality Server** ........................... 5  
  Fundamentals .................................................................................... 5  
  DataFlux Jobs and Services .............................................................. 7  
  Load and Unload Locales .................................................................. 9  
  Schemes .......................................................................................... 9  
  Create Match Codes .......................................................................... 10  
  Clusters .......................................................................................... 11  
  Sensitivity ....................................................................................... 12

**Chapter 3 • Locale Definitions** ................................................................... 15  
  Locale Definitions ............................................................................ 15

**Chapter 4 • DMSRVADM Procedure** ............................................................ 21  
  Overview: DMSRVADM Procedure .................................................. 21  
  Syntax: DMSRVADM Procedure ...................................................... 21  
  The Job Status Data Set .................................................................. 22  
  Security .......................................................................................... 23  
  Examples: DMSRVADM Procedure .................................................. 24

**Chapter 5 • DMSRVDATASVC Procedure** .................................................... 25  
  Overview: DMSRVDATASVC Procedure ........................................... 25  
  Syntax: DMSRVDATASVC Procedure ............................................... 25  
  The Input and Output Data Sets ...................................................... 28  
  Examples: DMSRVDATASVC Procedure ............................................ 29

**Chapter 6 • DMSRVPROCESSSVC Procedure** ............................................... 31  
  Overview: DMSRVPROCESSSVC Procedure ...................................... 31  
  Syntax: DMSRVPROCESSSVC Procedure ......................................... 31  
  The Input and Output Data Sets ...................................................... 34  
  Example: Run a DataFlux Data Management Service ..................... 34

**Chapter 7 • DQLOCLST Procedure** .............................................................. 35  
  Overview: DQLOCLST ..................................................................... 35  
  Syntax: DQLOCLST Procedure ....................................................... 35  
  Example: Create a Data Set of Locales ............................................ 36

**Chapter 8 • DQMATCH Procedure** .............................................................. 37  
  Overview: DQMATCH Procedure ..................................................... 37  
  Syntax: DQMATCH Procedure ........................................................ 38  
  Examples: DQMATCH Procedure ..................................................... 43
Chapter 9 • DQSCHEME Procedure ................................................................. 53
  Overview: DQSCHEME Procedure .............................................................. 53
  Syntax: DQSCHEME Procedure ................................................................. 54
  Using PROC DQSCHEME .......................................................................... 60
  Examples: DQSCHEME Procedure .............................................................. 62

Chapter 10 • AUTOCALL Macros ................................................................. 69
  Dictionary ...................................................................................................... 69

Chapter 11 • Functions and CALL Routines .............................................. 73
  Overview ...................................................................................................... 74
  Functions Listed Alphabetically ................................................................. 74
  Functions Listed by Category ..................................................................... 76
  Dictionary ...................................................................................................... 80

Chapter 12 • SAS Data Quality Server System Options ............................. 129
  SAS Data Quality Server System Options ................................................. 129
  Dictionary ...................................................................................................... 130

Appendix 1 • Deprecated Language Elements ......................................... 133
  Dictionary ...................................................................................................... 133

  Recommended Reading .............................................................................. 137
  Glossary ....................................................................................................... 139
  Index ............................................................................................................. 143
What's New in SAS 9.4 Data Quality Server

Overview

SAS 9.4 Data Quality Server includes the following enhancements:

- Interoperates with SSL-enabled DataFlux Data Management Server
- Lists the locales in the QKB using the new DQLOCLIST procedure
- Synchronizes results with DataFlux Data Management Studio

Interoperates with SSL-enabled DataFlux Data Management Server

In SAS 9.4M4, SAS Data Quality Server interoperates with SSL-enabled DataFlux Data Management Server 2.1 and later. Relevant language elements in SAS Data Quality Server can now use HTTPS URLs to communicate with the secured server software to run jobs and services.

Lists the Locales in the QKB Using the New DQLOCLIST Procedure

In SAS 9.4M4, SAS Data Quality Server added the DQLOCLST procedure. The DQLOCLST procedure creates a data set that includes the list of locales in the Quality Knowledge Base that is named in the system option DQSETUPLOC=.

Synchronizes Results with DataFlux Data Management Studio

In SAS 9.4M3, SAS Data Quality Server was upgraded to synchronize results with DataFlux Data Management Studio 2.7 or later.
Chapter 1
Overview of SAS 9.4 Data Quality Server

Capabilities and Requirements
SAS Data Quality Server consists of SAS language elements that perform data quality operations (matching, standardization, and so on). Other language elements interoperate with DataFlux Data Management Server to run jobs and services.

SAS Data Quality Server is delivered with a sample Quality Knowledge Base (QKB). As part of the installation and subsequent maintenance, you will replace the sample QKB with the latest QKB available from the SAS Downloads site.

SAS Data Quality Server communicates with DataFlux data management software to manage data assets through data quality, data integration, master data management (MDM), and federated data access. When the DataFlux Data Management Server is running in secure mode (with SSL enabled), it communicates only with clients that are also in secure mode. Communication in secure mode between SAS Data Quality Server and DataFlux Data Management Server requires the installation and configuration of OpenSSL 1.0 or higher.

In the Windows operating environment, OpenSSL must be installed in the Windows system directory or in a directory that is in the system PATH.

In the UNIX and Linux operating environments, OpenSSL must be installed in one of the following host directories:

- AIX: openssl.base
- HP-UX: openssl
- Linux: openssl
- Solaris 10: SUNWopenssl-libraries
- Solaris 11: library/security/openssl
Figure 1.1 SAS Data Quality Server Interactions with Data Management Software

Integration

Proper interoperability between SAS Data Quality Server and the data management software is contingent on several key components being installed or configured (or both).

- **License:** You must have the correct license to be able to use SAS 9.4 Data Quality Server with DataFlux Data Management Platform. For information, see SAS Support: License Assistance.

- **System Option Specifications:** SAS Data Quality Server is delivered with server and process language elements. Depending on your implementation and your operating system, you might need to rely on the system option elements to configure communication parameters with the Quality Knowledge Base.

- **Current Quality Knowledge Base:** The QKB should be updated each time a new one is released and made available through the SAS Downloads site.

- **Session and Definition Specifications:** You must make sure that your SAS Session is properly configured (that is, specify all options and file locations as needed) for SAS Data Quality Server. In addition, if necessary, you must specify definitions in your SAS data cleansing program(s).
The following resources have changed in recent releases:

- **SAS Support site** for both SAS and SAS DataFlux product support and resources. The MyDataFlux Portal is no longer supported.

- **SAS Downloads site** to download SAS DataFlux products, data updates, and Quality Knowledge Bases.

- **SAS Documentation site** and **SAS Install Center** to find current documentation for both SAS and SAS DataFlux products.

- **SAS Administrative Services** to manage your account, including how you can contact Contracts Support to obtain or renew a license.

- **SAS Support** to submit questions or a request for technical support.
Chapter 2
Getting Started with SAS Data Quality Server

Fundamentals

Specifying the Quality Knowledge Base ........................................ 5
Configure Your SAS Session for Data Quality ............................. 6
Specify Definitions in SAS Data Cleansing Programs .................. 6
Considerations for Installing and Updating the Software ............... 6
Access Source Tables in SAS Cloud Analytic Services ................. 7

DataFlux Jobs and Services ......................................................... 7
Creating Jobs and Services and Writing Information to the SAS Log 7
Running Jobs and Services on a DataFlux Data Management Server 7
DataFlux Data Management Server Passwords ............................ 8

Load and Unload Locales ............................................................ 9

Schemes .................................................................................... 9
Transport the Values of a Character Variable ............................... 9

Create Match Codes .................................................................... 10
Overview ............................................................................... 10
How to Create a Match Code .................................................... 11
Match Code Length .................................................................. 11

Clusters ..................................................................................... 11
Householding with the DQMATCH Procedure ............................ 12
Clustering with Exact Criteria ................................................... 12

Sensitivity .................................................................................. 12

Fundamentals

Specifying the Quality Knowledge Base

To access a Quality Knowledge Base, SAS programs reference the value set in the DQSETUPLOC= system option. The value of the system option is the path to the root directory of a Quality Knowledge Base.

- If you move your Quality Knowledge Base, update the path specifications with DQSETUPLOC= accordingly.
- If your site uses multiple Quality Knowledge Bases, reference each location with the DQSETUPLOC= option.
Configure Your SAS Session for Data Quality

Use the DQOPTIONS= system option to configure your SAS session for data quality. Specify the option in your SAS start-up command. SASV9.CFG. DQOPTIONS= enables you to specify one or more option-value pairs that pertain to the operation of the SAS Data Quality Server software.

The DQOPTIONS= system option enables two option-value pairs. The DQSRVPROTOCOL=WIRELINE pair improves the performance of the DMSRVDATASVC and DMSRVPROCESSSVC procedures by streamlining data transfer to and from the DataFlux Data Management Server. It is the default on z/OS, and improves performance on all other platforms. In other operating environments, the default Simple Object Access Protocol (SOAP) is recommended.

TRANSCODE=IGNORE | WARN specifies that transcoding errors between character sets are to be ignored, and SAS processing is allowed to continue. By default, transcoding errors terminate SAS processing.

See “DQOPTIONS= System Option” on page 131 for additional information.

Specify Definitions in SAS Data Cleansing Programs

To specify definitions in your SAS data cleansing programs, you need to know the names of the definitions that are available in a particular locale. To find out the names of the definitions that are available in a particular local, use the “%DQPUTLOC AUTOCALL Macro” on page 70.

To display information about a locale that is currently loaded into memory, use the “DQLOCALEINFOGET Function” on page 102.

To display a list of definitions in a specified locale, use the “DQLOCALEINFOLIST Function” on page 102 to return the name of the locale that best fits your data.

Considerations for Installing and Updating the Software

The SAS Data Quality Server software is delivered with a sample Quality Knowledge Base. After you install the SAS Data Quality Server software, download the latest Quality Knowledge Base, including your choice of locales, from the SAS Downloads site.

To maximize performance, download new Quality Knowledge Bases as they are made available on the SAS Downloads site. Check the latest release notes to determine whether definitions used by your jobs have been updated. Decide whether you need any of the new locale support that might have been added. (If you decide that you need new locale support, please contact SAS to revise your license agreement.) For a listing of the latest enhancements to the QKB, refer to the “What’s New in DataFlux Quality Knowledge Base” in the QKB Help.

When you update your Quality Knowledge Base, you might want to install it in a new location rather than overwriting your existing Quality Knowledge Base. This decision is particularly important if you have customized your Quality Knowledge Base in the previous release. Customizations are made with the DataFlux Data Management Studio software. If you install your updated Quality Knowledge Base in a new location, make sure you reference the Quality Knowledge Base by specifying the appropriate value in the DQSETUPLOC= system option.
If you customized your previous Quality Knowledge Base, evaluate those changes and carry them over to your new Quality Knowledge Base as needed.

**CAUTION:**

When you upgrade your Quality Knowledge Base, be sure to regenerate your existing match codes so that they are consistent with the newly created match codes.

---

**Access Source Tables in SAS Cloud Analytic Services**

SAS 9.4 data quality programs running on SAS Workspace Servers can access data on SAS Cloud Analytic Services, as described in the following documents:

- For conceptual information about SAS Cloud Analytic Services, see *SAS Cloud Analytic Services: Fundamentals*.
- For syntax and general information about CAS-specific language elements, see *SAS Cloud Analytic Services: Language Reference*.

---

**DataFlux Jobs and Services**

**Creating Jobs and Services and Writing Information to the SAS Log**

Jobs and services that are run on a DataFlux Data Management Server fulfill separate needs. Use jobs to access larger data sets in batch mode when a client application is not waiting for a response. Use services and small data sets in real time when clients await a response from the server.

To create jobs and services for your DataFlux Data Management Server, use the DataFlux Data Management Studio application. You can also use DataFlux Data Management Studio to create jobs to analyze the quality of your data. The DMSRVPROFILEJOB function generates a profile. The DMSRVBATCHJOB function runs data jobs as well as process jobs.

Use DataFlux Data Management Studio software to create jobs and services using a drag-and-drop interface. You can trigger the DataFlux Data Management Studio jobs with the function DMSRVBATCHJOB. You can run DataFlux Data Management services with the DMSRVDATASVC and DMSRVPROCESSSVC procedures.

Jobs and services that run on the DataFlux Data Management Servers generate information that is written to the log. You can copy server logs using the DMSRVCOPYLOG function. Based on the information returned by the DMSRVJOBSTATUS function, you can terminate jobs using the DMSRVKILLJOB function.

---

**Running Jobs and Services on a DataFlux Data Management Server**

Follow these steps to run jobs and services on a DataFlux Data Management Server.

1. For any client session using Wireline and in the z/OS operating environment, for DataFlux Data Management Servers, configure your DataFlux Data Management Server to use the Wireline protocol. This is described in the *DataFlux Data Management Server: User's Guide*. The Wireline protocol improves data transfer performance.
2. In the z/OS operating environment, ensure that your SAS session is configured to use the Wireline protocol. This is the default setup and can be manually configured as described in “Configure Your SAS Session for Data Quality” on page 6.

3. Create jobs and services using the DataFlux Data Management Studio software.

4. Upload the jobs to the DataFlux Data Management Server using the DataFlux Data Management Server Manager.

5. Create and run the SAS programs that execute or trigger the jobs and services on the DataFlux Data Management Server.

To run jobs and services, you do not need to load a Quality Knowledge Base onto your local host. The DataFlux Data Management Server handles all interactions with your Quality Knowledge Bases.


**DataFlux Data Management Server Passwords**

If security is implemented on your DataFlux Data Management Server, include user names and passwords in the procedures and function calls that access that server. Specify the passwords directly, in plain text, or as encoded passwords. SAS recognizes encoded passwords and decodes them before it sends the passwords to the DataFlux Data Management Server.

This example shows how to encode a password and use that password in a call to the DMSRVDATASVC procedure:

```sas
/*   Encode password in file. */
filename pwfile 'c:\dataEntry01Pwfile';
proc pwencode in='Oe3s2m5' out=pwfile;
run;

/* Load encoded password into macro variable. */
data _null_;
  infile pwfile obs=1 length=l;
  input @;
  input @1 line $varying1024. l;
  call symput ('dbpass', substr(line,1,l));
run;

/*  Run service on secure DataFlux Data Management Server  */
proc dmsrvdatasvc
  service='cleanseCorpName' host='entryServer1'
  userid='DataEntry1'        password="&dbpass"
  data=corpName              out=corpNameClean;
run;
```

PROC PWENCODing concepts, syntax, and examples are documented in the Base SAS Procedures Guide.
Load and Unload Locales

You need to load and unload locales to run data-cleansing programs in SAS. Conversely, you do not need to load locales if your SAS programs run jobs and services on a DataFlux Data Management Server.

Before you run data-cleansing programs in SAS, load locales into memory as described in "%DQLOAD AUTOCALL Macro" on page 69. Use the DQSETUPLOC= option to specify the location of the Quality Knowledge Base. Use the DQLOCALE= option to specify an ordered list of locales.

The ordered list of locales is pertinent only when one of the following conditions is true:

- A locale is not specified by name.
- The specified locale is not loaded into memory.
- Input data is insufficient for the DQLOCALEGUESS function.

If a locale cannot be established, SAS searches the list of locales. SAS references the first definition it finds that has the specified name.

You can change the values of the system options DQSETUPLOC= and DQLOCALE=. However, doing so does not load different locales into memory. For this reason, it is recommended that you use the "%DQLOAD AUTOCALL Macro" on page 69 to change the values of the two data quality system options.

If you customize your Quality Knowledge Base, you must reload macros into memory with the %DQLOAD AUTOCALL macro before cleansing data.

After you submit your data-cleansing programs, you can unload the locale from memory by using the "%DQUNLOAD AUTOCALL Macro" on page 72.

New locales and updates to existing locales are provided periodically by SAS DataFlux in the form of a new Quality Knowledge Base. Quality Knowledge Bases that you have licensed can be downloaded from the SAS Downloads site.

Schemes

Transport the Values of a Character Variable

A scheme is a file that you create to transform the values of a character variable. Applying the scheme to the data transforms similar representations of a data value into a standard representation.

To create and apply multiple schemes in a single procedure, see Chapter 9, “DQSCHEME Procedure,” on page 53. Use the information found in “DQSCHEMEAPPLY Function” on page 117 and “DQSCHEMEAPPLY CALL Routine” on page 120 to apply schemes as well.

This information pertains only to schemes that are QKB (DataFlux) types of schemes. The Scheme Builder application does not recognize schemes that are stored as SAS DATA sets.
Create Match Codes

Overview

Match codes are encoded representations of character values that are used for analysis, transformation, and standardization of data. Use the following procedures and functions to create match codes:

The DQMATCH procedure
creates match codes for one or more variables or parsed tokens that have been extracted from a variable. The procedure can also assign cluster numbers to values with identical match codes. See Chapter 8, “DQMATCH Procedure,” on page 37 for additional information.

The DQMATCH Function
generates match codes for tokens that have been parsed from a variable. See “DQMATCH Function” on page 104 for additional information.

The DQMATCHPARSED Function
See “DQMATCHPARSED Function” on page 106 for additional information.

Match codes are created by the DQMATCH procedure and by the DQMATCH and DQMATCHPARSED functions. The functions DQMATCH and DQMATCHPARSED return one match code for one input character variable. With these tools, you can create match codes for an entire character value or a parsed token extracted from a character value.

• During processing, match codes are generated according to the specified locale, match definition, and sensitivity-level.
• The locale identifies the language and geographical region of the source data. For example, the locale ENUSA specifies that the source data uses the English language as it is used in the United States of America.
• The match definition in the Quality Knowledge Base identifies the category of the data and determines the content of the match codes. Examples of match definitions are named ADDRESS, ORGANIZATION, and DATE (YMD).

To determine the match definitions that are available in a Quality Knowledge Base, consult the Help for that QKB. Alternatively, use the DQLOCALEINFOLIST function to return the names of the locale’s match definitions. Use the DQLOCALEINFOLIST function if one (or both) of the following statements are true:

• Your site has added definitions to your Quality Knowledge Base.
• Your site has modified the default Quality Knowledge Base using the Customize software in DataFlux Data Management Studio.

The sensitivity level is a value between 50 and 95 that determines the amount of information that is captured in the match code, as described in “Sensitivity” on page 12.

If two or more match codes are identical, a cluster number can be assigned to a specified variable, as described in “Clusters” on page 11.

The content of the output data set is determined by option values. You can include values that generate unique match codes, and you can include and add a cluster number to blank or missing values. You can also concatenate multiple match codes.
Match codes are also generated internally when you create a scheme with the DQSCEME procedure, as described in “Schemes” on page 9. Match codes are also created internally by the DQSCEMEAPPLY function and the DQSCEMEAPPLY CALL routine. The match codes are used in the process of creating or applying a scheme.

**How to Create a Match Code**

You can create two types of match codes:

- Simple match codes from a single input character variable.
- A concatenation of match codes from two or more input character variables. The separate match codes are concatenated into a composite match code.

Use the DELIMITER= option to specify that a delimiter exclamation point (!) is to be inserted between the simple match codes in the combined match code.

To create simple match codes, specify one CRITERIA statement, one input variable identified in the VAR= option, and one output variable identified with the MATCHCODE= option.

Composite match codes are similar, except that you specify multiple CRITERIA statements for multiple variables. All the CRITERIA statements specify the same output variable in their respective MATCHCODE= options.

SAS Data Quality Server software creates match codes using these general steps:

1. Parse the input character value to identify tokens.
2. Remove insignificant words.
3. Remove some of the vowels. Remove fewer vowels when a scheme-build match definition has been specified.
4. Standardize the format and capitalization of words.
5. Create the match code by extracting the appropriate amount of information from one or more tokens, based on the specified match definition and level of sensitivity.

Certain match definitions skip some of these steps.

*Note:* To analyze or join two or more data sets using match codes, create the match codes in each data set with identical sensitivity levels and match definitions.

**Match Code Length**

Match codes can vary in length between 1 and 1024 bytes. The length is determined by the specified match definition. If you receive a message in the SAS log that states that match codes have been truncated, extend the length of the match code variable. Truncated match codes do not produce accurate results.

**Clusters**

Clusters are numbered groups of values that generate identical match codes or that have an exact match of characters. Clusters are used in the creation of schemes using the DQSCEME procedure. The cluster with the greatest number of members becomes the transformation value for the scheme.
Householding with the DQMATCH Procedure

You can use the DQMATCH procedure to generate cluster numbers as it generates match codes. An important application for clustering is commonly referred to as householding. Members of a family or household are identified in clusters that are based on multiple criteria and conditions.

To establish the criteria and conditions for householding, use multiple CRITERIA statements and CONDITION= options within those statements.

- The integer values of the CONDITION= options are reused across multiple CRITERIA statements to establish groups of criteria.
- Within each group, match codes are created for each criteria.
- If a source row is to receive a cluster number, all of the match codes in the group must match all of the codes in another source row.
- The match codes within a group are therefore evaluated with a logical AND.

If more than one condition number is specified across multiple CRITERIA statements, there are multiple groups and multiple groups of match codes. In this case, source rows receive cluster numbers when any groups match any other group in another source row. The groups are therefore evaluated with a logical OR.

For an example of householding, assume that a data set contains customer information. To assign cluster numbers, you use two groups of two CRITERIA statements. One group (condition 1) uses two CRITERIA statements to generate match codes based on the names of individuals and an address. The other group (condition 2) generates match codes based on organization name and address. A cluster number is assigned to a source row when either pair of match codes matches at least one group that matches the match codes from another source row. The code and output for this example are provided in “Example 5: Clustering with Multiple CRITERIA Statements” on page 49.

Clustering with Exact Criteria

Use the EXACT= option of the DQMATCH procedure's CRITERIA statement to use exact character matches as part of your clustering criteria. Exact character matches are helpful in situations where you want to assign cluster numbers using a logical AND of an exact number and the match codes of a character variable.

For example, you could assign cluster numbers using two criteria: one using an exact match on a customer ID values and the other using a match code generated from customer names. The syntax of the EXACT= option is provided in Chapter 8, “DQMATCH Procedure,” on page 37.

Sensitivity

The amount of information contained in match codes is determined by a specified sensitivity level. Changing the sensitivity level enables you to change what is considered a match. Match codes created at lower levels of sensitivity capture little information about the input values. The result is more matches, fewer clusters, and more values in each cluster. See “Clusters” on page 11 for additional information.

Higher sensitivity levels require that input values are more similar to receive the same match code. Clusters are more numerous, and each cluster contains fewer entries. For
example, when collecting customer data that is based on account numbers, cluster on account numbers with a high sensitivity value.

In some data cleansing jobs, a lower sensitivity value is needed. To transform the following names to one consistent value using a scheme, specify a lower sensitivity level.

Patricia J. Fielding
Patty Fielding
Patricia Feelding
Patty Fielding

All four values are assigned to the same cluster. The clusters are transformed to the most common value, **Patty Fielding**.

Sensitivity values range from 50 to 95. The default value is 85.

To arrive at the sensitivity level that fits your data and your application, test with the DQMATCH procedure. Alternatively, create analysis data sets with the DQSCHEME procedure.
Parse definitions are referenced when you want to create parsed input values. Parsed input values are delimited so that the elements in those values can be associated with named tokens. After parsing, specific contents of the input values can be returned by specifying the names of tokens.

Parse definitions and tokens are referenced by the following routine and functions:

- “DQPARSE CALL Routine” on page 109
- “DQPARSEINFOGET Function” on page 110
- “DQTOKEN Function” on page 126
- “DQPARSETOKENGET Function” on page 113
- “DQPARSETOKENPUT Function” on page 114

Parsing a character value assigns tokens only when the content in the input value meets the criteria in the parse definition. Parsed character values can therefore contain empty tokens. For example, three tokens are empty when you use the DQPARSE function to parse the character value **Ian M. Banks**. When using the NAME parse definition in the ENUSA locale, the resulting token/value pairs are as follows:

NAME PREFIX
   empty

GIVEN NAME
   Ian
Global Parse Definitions

Global parse definitions contain a standard set of parse tokens that enable the analysis of similar data from different locales. For example, the ENUSA locale and the DEDEU locale both contain the parse definition ADDRESS (GLOBAL). The parse tokens are the same in both locales. This global parse definition enables the combination of parsed character data from multiple locales.

All global parse definitions are identified by the (GLOBAL) suffix.

Extraction Definitions

Extraction definitions extract parts of an input string and assign them to corresponding tokens of the associated data type. Extraction input values are delimited so that the elements in those values can be associated with named tokens. After extraction, specific contents of the input values can be returned by specifying the names of tokens.

Extraction definitions and tokens are referenced by the following functions:

- “DQEXTRACT Function” on page 93
- “DQEXTINFOGET Function” on page 92
- “DQEXTTOKENGET Function” on page 94
- “DQEXTTOKENPUT Function” on page 95

Extracting a character value assigns tokens when the content in the input value meets the criteria in the extraction definition. For example, using the string "100 Slightly used green Acme MAB-6200 telephone $100 including touch-tone buttons" as input results in the following output mapping between tokens and substrings.

QUANTITY
100

BRAND
“ACME”

MODEL
“MAB-6200”

COLOR
“green”

PRICE
“$100”
DESCRIPTION
“slightly used telephone including touch-tone buttons”

Extracted character values can also contain empty tokens. For example, in the illustration above, if the input string did not contain a price, then PRICE would contain an “empty” token.

Match Definitions
Match definitions are referenced during the creation of match codes. Match codes provide a variable method of clustering similar input values as a basis for data cleansing jobs such as the application of schemes.

When you create match codes, you specify a sensitivity level. The default sensitivity level is specified by the procedure or function, rather than the match definition.

Match definitions are referenced by the following procedures and functions:
- Chapter 8, “DQMATCH Procedure,” on page 37
- Chapter 9, “DQSCHEME Procedure,” on page 53
- “DQMATCH Function” on page 104
- “DQMATCHINFOGET Function” on page 105
- “DQMATCHPARSED Function” on page 106

When you create match codes for parsed character values, your choice of match definition depends on the parse definition that was used to parse the input character value. To determine the parse definition that is associated with a given match definition, use the “DQMATCHINFOGET Function” on page 105.

Note: For match definitions that work with dates, such as DATE (MDY) in the ENUSA locale, input values must be character data rather than SAS dates.

Case and Standardization Definitions
Case and standardization definitions are applied to character values to make them more consistent for the purposes of display or in preparation for transforming those values with a scheme.

Case definitions are referenced by the “DQCASE Function” on page 91. Standardization definitions are referenced by the “DQSTANDARDIZE Function” on page 125.

Case definitions transform the capitalization of character values. For example, the case definition Proper in the ENUSA locale takes as input any general text. It capitalizes the first letter of each word, and uses lowercase for the other letters in the word. It also recognizes and retains or transforms various words and abbreviations into uppercase. Other case definitions, such as PROPER – ADDRESS, apply to specific text content.

Standardization definitions standardize the appearance of specific data values. In general, words are capitalized appropriately based on the content of the input character values. Also, adjacent blank spaces are removed, along with unnecessary punctuation. Additional standardizations might be made for specific content. For example, the standardization definition STATE (FULL NAME) in the locale ENUSA converts abbreviated state names to full names in uppercase.
Standardization of Dates in the EN Locale

In the EN locale, dates are standardized to two-digit days (00–31), two-digit months (01–12), and four-digit years. Input dates must be character values rather than SAS dates.

Spaces separate (delimit) the days, months, and years, as shown in the following table:

<table>
<thead>
<tr>
<th>Input Date</th>
<th>Standardization Definition</th>
<th>Standardized Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>July04, 03</td>
<td>Date (MDY)</td>
<td>07 04 2003</td>
</tr>
<tr>
<td>July 04 04</td>
<td>Date (MDY)</td>
<td>07 04 1904</td>
</tr>
<tr>
<td>July0401</td>
<td>Date (MDY)</td>
<td>07 04 2001</td>
</tr>
<tr>
<td>04.07.02</td>
<td>Date (DMY)</td>
<td>04 07 2002</td>
</tr>
<tr>
<td>04-07-2004</td>
<td>Date (DMY)</td>
<td>04 07 2004</td>
</tr>
<tr>
<td>03/07/04</td>
<td>Date (YMD)</td>
<td>2003 07 04</td>
</tr>
</tbody>
</table>

Two-digit year values are standardized as follows:

• If an input year is greater than 00 and less than or equal to 03, the standardized year is 2000, 2001, 2002, or 2003.

• Two-digit input year values that are greater than or equal to 04 and less than or equal to 99 are standardized into the range of 1904–1999.

For example, an input year of 03 is standardized as 2003. An input year of 04 is standardized as 1904. These standardizations are not affected by the value of the SAS system option YEARCUTOFF=.

Gender Analysis, Locale Guess, and Identification Definitions

Gender analysis, locale guess, and identification definitions enable you to make determinations about character values. With these definitions, you can determine the following:

• the gender of an individual based on a name value
• the locale that is the most suitable for a given character value
• the category of a value, which is chosen from a set of available categories

Gender analysis definitions determine the gender of an individual based on that individual’s name. The gender is determined to be unknown if the first name is used by both males and females. If no other clues are provided in the name, or if conflicting clues are found, gender analysis definitions are referenced by the “DQGENDER Function” on page 96.

Locale guess definitions allow the software to determine the locale that is most likely represented by a character value. All of the locales that are specified in the value of the
DQLOCALE= system option are considered, but only if they contain the specified locale guess definition. Locale guess definitions are referenced by the “DQLOCALEGUESS Function” on page 100.

Identification definitions are used to categorize character values. For example, using the Entity identification definition in the ENUSA locale, a name value can apply to an individual or an organization. Identification definitions are referenced by the “DQIDENTIFY Function” on page 99.

**Pattern Analysis Definitions**

Pattern analysis definitions enable you to determine whether an input character value contains characters that are alphabetic, numeric, non-alphanumeric (punctuation marks or symbols), or a mixture of alphanumeric and non-alphanumeric. The ENUSA locale contains two pattern analysis definitions: WORD and CHARACTER.

The pattern analysis definition WORD is referenced by the DQPATTERN function. This generates one character of analytical information for each word in the input character value. See “DQPATTERN Function” on page 116 for additional information. The CHARACTER definition generates one character of analytical information for each character in the input character value.
Overview: DMSRVADM Procedure

What Does the DMSRVADM Procedure Do?

The DMSRVADM procedure creates a data set that provides the name, type, and description of all DataFlux Data Management jobs. This includes jobs that ran or that are running on a specified port on a DataFlux Data Management Server. Status information is provided for all jobs that have a log file on the server.

Syntax: DMSRVADM Procedure

PROC DMSRVADM

< HOST=host-name>
<OUT=output-data-set>
<PASSWORD,password>
<PORT=job-port-number>
<USERID=user-id>;

PROC DMSRVADM Statement

The DMSRVADM procedure creates a status information data set.
Syntax

PROC DMSRVADM
  <HOST=host-name>
  <OUT=output-data-set>
  <PASSWORD=password>
  <PORT=job-port-number>
  <USERID=userid>;

Optional Arguments

HOST= host-name
 identifies the host of the DataFlux Data Management Server. The host name can be a
variable or a literal string. The literal string or the value of the variable is the URL of
the server in single quotation marks.

When DataFlux Data Management Server is secured with SSL, the URL must use
https instead of http (for example, https://myhost.unx.com).

Default localhost.

OUT= output-data-set
 specifies the storage location of the job status data set.

Default If the OUT= option is not specified, the input data set_LAST_ is used.

PASSWORD= password
 authenticates the user according to the registry in the DataFlux Data Management
Server. The password can be plain text or encoded in SAS.

Note If security has not been configured on the server, the PASSWORD= option is
ignored.

PORT= port-number
 identifies the port through which the host communicates with the DataFlux Data
Management Server.

Default If the port-number is not specified, or the value is 0, or a negative number,
the default port number 21036 is used.

USERID= userid
 authenticates the user according to the registry in the DataFlux Data Management
Server.

The Job Status Data Set

The job status data set contains the following variables:

JOBID
 specifies the job-id that was submitted to the DataFlux Data Management Server.
The job-id is previously set by a function such as DMSRVBATCHJOB. The job-id is
an argument in these functions:
• DMSRVCOPYLOG
• DMSRVDELETELOG
• DMSRVJOBSTATUS
• DMSRVKILLJOB

See “Functions Listed Alphabetically” on page 74 for a complete list of functions and call routines.

STATUS
specifies the following job status codes:

0
Job completed successfully.

1
Job failed.

2
Job still running.

Descriptions and associated job types are listed in the table below.

<table>
<thead>
<tr>
<th>JOBDESC</th>
<th>JOBTYPEx</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Service</td>
<td>0</td>
</tr>
<tr>
<td>Data Job</td>
<td>1</td>
</tr>
<tr>
<td>Profile Job</td>
<td>2</td>
</tr>
<tr>
<td>Process Job/Service</td>
<td>3</td>
</tr>
<tr>
<td>Repository Job</td>
<td>4</td>
</tr>
<tr>
<td>Unknown Type</td>
<td>Undetermined</td>
</tr>
</tbody>
</table>

Security

If security is implemented on a DataFlux Data Management Server, then you might need to register user name and password credentials with the function DMSRVUSER before you run PROC DMSRVADM.

See “DMSRVUSER Function” on page 89.
Examples: DMSRVADM Procedure

Example 1: Generate a Job Status Data Set

This example generates a data set that provides information about jobs that are running or have run on a DataFlux Data Management Server. The job status data set contains information about jobs that are represented by log files on the server.

```
proc dmsrvadm
   out=work.jobReport
   host='http://myhost.unx.com'   port=50001;
run;
```

Example 2: Clean Up Jobs and Logs

This example generates a job report and then uses the contents of the report to terminate all jobs and delete all log files on the DataFlux Data Management Server:

```
proc dmsrvadm
   out=work.jobReport
   host='http://myhost.unx.com'   port=50001;
run;

data _null_;  
set work.joblist;
   kjrc=dmsrvkilljob (jobid, 'http://myhost.unx.com', 50001);
   dlrc=dmsrvdeletelog (jobid, 'http://myhost.unx.com', 50001);
run;
```
Overview: DMSRVDATASVC Procedure

What Does the DMSRVDATASVC Procedure Do?

The DMSRVDATASVC procedure runs a DataFlux Data Management Studio real-time service on a DataFlux Data Management Server. DataFlux Data Management real-time services are batch processes that are intended to cleanse smaller amounts of data at the point of data entry. Data processing is intended to be synchronous, when a client application requests the service and awaits a response. The DMSRVDATASVC procedure authenticates you on the server, requests a service, delivers input data to the server, and delivers output data to a SAS DATA set.

To improve performance, large input data sets are delivered to the DataFlux Data Management Server in chunks of a specified size.

To cleanse or analyze larger amounts of data asynchronously, execute a DataFlux job using the function DMSRVBATCHJOB.

Syntax: DMSRVDATASVC Procedure

See: The documentation for your operating environment for specific DMSRVDATASVC procedure information.
PROC DMSRVDATASVC
<DATA=input-data-set>
<BLOCKSIZE=rows-per-message>
<HOST=host-name>
<PARMLIST=parameter-list>
<MISSINGVARSOK>
<NOPRINT>
<OUT=output-data-set>
<PASSWORD=password-on-server>
<PORT=port-number>
<SERVICE=service-name>
<SERVICEINFO>
<TIMEOUT=message-processing-limit>
<TRIM>
<USERID=user-name-on-server>;

PROC DMSRVDATASVC Statement
The DMSRVDATASVC procedure runs a DataFlux Data Management Studio real-time service on a DataFlux Data Management Server.

Syntax
PROC DMSRVDATASVC
<DATA=input-data-set>
<BLOCKSIZE=rows-per-message>
<HOST=host-name>
<PARMLIST=parameter-list>
<MISSINGVARSOK>
<NOPRINT>
<OUT=output-data-set>
<PASSWORD=password-on-server>
<PORT=port-number>
<SERVICE=service-name>
<SERVICEINFO>
<TIMEOUT=message-processing-limit>
<TRIM>
<USERID=user-name-on-server>;

Actions
BLOCKSIZE= rows-per-message
specifies the number of rows of source data that are transmitted to the DataFlux Data Management Server, in multiple messages. If this option is not specified, then the entire data set is transmitted in a single message. Transmitting large data sets in a single message can restrict resources on the DataFlux Data Management Server. The server processes each message separately. Output is delivered as usual in a single message.

The DataFlux Data Management Studio program needs to be written to accommodate multiple messages.
Restriction Services that require the entire data set, such as those that calculate averages or frequencies, cannot use the BLOCKSIZE= option.

**DATA= data-set-name**
identifies name of the input data set.

**Default** If the DATA= option is not specified, the input data set _LAST_ is used.

**HOST= host-machine**
identifies the host of the DataFlux Data Management Server. The host name can be a variable or a literal string. The literal string or the value of the variable is the URL of the server in single quotation marks.

When DataFlux Data Management Server is secured with SSL, the URL must use https instead of http (for example, https://myhost.unx.com).

**Default** localhost.

**PARMLIST= parameter-list**
takes a quoted string as its value. The contents of the string is a series of name-value pairs. These pairs are passed to the service. If the service uses a parameter list with a name that matches a name in the parameter-list, then the name is assigned the corresponding value in the parameter-list.

In the parameter-list, both the name and value in each pair must appear within single quotation marks. An equal sign must separate the name and value. A comma separates each name-value pair from the next name-value pair in the list.

**MISSINGVARSOK**
indicates that the DataFlux real-time service is to be allowed to continue to run when one or more variables (or table columns) are missing from the input data set. When the MISSINGVARSOK option is set, any data that is missing from the input data set is assumed to be non-critical, or required by the DataFlux real-time service.

**Default** MISSINGVARSOK is not set by default.

**NOPRINT**
if the SERVICEINFO option is specified, suppresses writing the SERVICEINFO information to the SAS log.

**OUT= output-data-set**
identifies the name of the output data set. DataFlux Data Management Studio services always create new data sets or overwrite existing data sets.

**Default** If the OUT= option is not specified, the input data set _LAST_ is used.

**PASSWORD= password**
authenticates the user according to the registry in the DataFlux Data Management Server. The password can be plain text or encoded in SAS.

**Note** If security has not been configured on the server, the PASSWORD= option is ignored.

**PORT= port-number**
identifies the port number through which the host communicates with the DataFlux Data Management Server. If this option is not specified, or if the value is zero or a negative number, the default port number 21036 is used.
**SERVICE**= service-name

Identifies the service on the DataFlux Data Management Server.

**SERVICEINFO**

Writes the input and output columns used by the given service to the data set specified by the OUT= option.

The data set has four columns:

- Name is the column name.
- Type is the type of data in column - character(C) or numeric(N).
- Length is the length of column data.
- Class is the input, output, or macro.

Default: The service information is also written to the SAS log.

Restriction: If SERVICEINFO is specified, the service is not run. Any options related to the execution of the service, such as the BLOCKSIZE= option, are ignored.

**TIMEOUT**= message-processing-limit

Specifies a time in seconds after which the procedure terminates if the localhost has not received a response from the DataFlux Data Management Server. If data is delivered to the server in multiple messages using the BLOCKSIZE= option, the TIMEOUT= value is applied to each message.

Tip: A value of zero or a negative number enables the procedure to run without a time limit.

**TRIM**

Removes any blank spaces from the end of the input values.

Default: TRIM is not set by default.

**USERID**= user-name

Identifies the user according to the registry in the DataFlux Data Management Server.

Note: If security has not been configured on the server, the USERID= option is ignored.

---

**The Input and Output Data Sets**

**The Input Data Set**

The DMSRVDATASVC procedure acquires the names of the columns that the service expects to receive as input from the DataFlux Data Management service. In this case, the name of the input data set must match the name that is specified in the service. If the
expected column names do not match the column names in the input data set, then the DMSRVDATASVC procedure terminates.

Services can also be created where any named data set can be used for input. In this case, there is no input data set name required.

**The Output Data Set**

If the output data set exists, new output data overwrites any existing data. The type of the output data is determined by the service.

### Examples: DMSRVDATASVC Procedure

#### Example 1: Send Input Data to Service and Output to Data Set

This example reads in a data file to a specified service, and the output from the data service appears in the indicated output data set. The HOST= option names the server, and the PORT= option specifies the port as the default, so the server communicates over port 21036 of ‘myserver’. The SERVICE= option is the service to which the input data is sent, and the DATA= option specifies the input data set. The OUT= option specifies the output data set. The TIMEOUT= option is the length of time, in seconds, that the job is allowed to run before being terminated. The USERID= and PASSWORD= options are the user credentials under which the job is being executed. The SERVICE was previously created and uploaded to the DataFlux Data Management Server.

```c
/* send input data dqsio.dfsample to service analysis.ddf. */
/* output from the data service appears in data set work.outsrv17 */
PROC DMSRVDATASVC
   HOST='http://myhost.unx.com'   PORT=21036
   SERVICE='analysis.ddf'
   DATA=dqsio.dfsample
   OUT=outsrv17
   TIMEOUT=360
   USERID='myname'
   PASSWORD='mypassword';
RUN;
```
Example 2: Run a DataFlux Data Management Studio Service

Features:

```
PROC DMSRVDATASVC
  SERVICE=
  DATA=
  OUT=
```

This example runs a DataFlux Data Management Studio service on a DataFlux Data Management Server that is installed on the local host. The PORT= option is not set, so the server communicates over the default port 21036. The DATA= option specifies the input data set. The OUT= option specifies the output data sets. The SERVICE was previously created and uploaded to the DataFlux Data Management Server.

```
PROC DMSRVDATASVC
  SERVICE='myService'
  DATA=work.insrv
  OUT=work.outsrv;
RUN;
```
Overview: DMSRVPROCESSSVC Procedure

What Does the DMSRVPROCESSSVC Procedure Do?

The DMSRVPROCESSSVC procedure runs a service on a DataFlux Data Management Server. The procedure executes a DM process service, or, when the SERVICEINFO= option is specified, it produces a list of inputs and outputs managed by the service.

A process service generates a list of name-value pairs on output. When the SERVICEINFO= option is specified, it produces a list of inputs and outputs managed by the service.

When the SERVICEINFO= option is specified, the procedure generates an output data set with two columns. The first column, named "Parameter", lists the parameter names. The second column, named "Type", lists the parameter as "INPUT" or "OUTPUT".

Syntax: DMSRVPROCESSSVC Procedure

PROC DMSRVPROCESSSVC
<HOST=host-name>

See: The documentation for your operating environment for specific DMSRVPROCESSSVC procedure information.
PROC DMSRVPROCESSSVC Statement

The DMSRVPROCESSSVC procedure runs a service on a DataFlux Data Management Server.

Syntax

PROC DMSRVPROCESSSVC
  <HOST=host-name>
  <PARMLIST=parameter-list>
  <NOPRINT>
  <OUT=output-data-set>
  <PASSWORD=password-on-server>
  <PORT=port-number>
  <SERVICE=service-name>
  <SERVICEINFO>
  <TIMEOUT=message-processing-limit>
  <USERID=user-name-on-server>;

Actions

HOST=host-machine

identifies the host of the DataFlux Data Management Server. The host name can be a variable or a literal string. The literal string or the value of the variable is the URL of the server in single quotation marks.

When DataFlux Data Management Server is secured with SSL, the URL must use https instead of http (for example, https://myhost.unx.com).

Default localhost.

PARMLIST=parameter-list

takes a quoted string as its value. The contents of the string is a series of name-value pairs. These pairs are passed to the service. If the service uses a parameter list with a name that matches a name in the parameter-list, then the name is assigned the corresponding value from the parameter-list.

In the parameter-list, both the name and value in each pair must appear within single quotation marks. An equal sign must separate the name and the value. A comma must separate each name-value pair from the next name-value pair.

NOPRINT

if the SERVICEINFO option is specified, suppresses writing the SERVICEINFO information to the SAS log.
OUT=output-data-set
identifies the name of the output data set. DataFlux Data Management real-time
services always create new data sets or overwrite existing data sets.

Default If the OUT= option is not specified, the input data set _LAST_ is used.

PASSWORD=password
authenticates the user according to the registry in the DataFlux Data Management
Server. The password can be plain text or encoded in SAS.

Note If security has not been configured on the server, the PASSWORD= option is
ignored.

PORT=port-number
identifies the port number through which the host communicates with the DataFlux
Data Management Server. If this option is not specified, or if the value is zero or a
negative number, the default port number 21036 is used.

Default 21036

SERVICE=service-name
identifies the service on the DataFlux Data Management Server.

SERVICEINFO
writes the input and output columns used by the given service to the data set
specified by the OUT= option.

The data set has four columns:
  • Name is the column name.
  • Type is the type of data in column -character(C) or numeric(N).
  • Length is the length of column data.
  • Class is the input, output, or macro.

Default The service information is also written to the SAS log.

Restriction If SERVICEINFO is specified, the service is not run. Any options
related to the execution of the service, such as the BLOCKSIZE= option, are ignored.

TIMEOUT=message-processing-limit
specifies a time in seconds after which the procedure terminates if the localhost
has not received a response from the DataFlux Data Management Server. If data is
delivered to the server in multiple messages using the BLOCKSIZE= option, the
TIMEOUT= value is applied to each message.

Tip A value of zero or a negative number enables the procedure to run without a
time limit.

USERID=user-name
identifies the user according to the registry in the DataFlux Data Management
Server.

Note If security has not been configured on the server, the USERID= option is
ignored.
The Input and Output Data Sets

The Input Data Set

The DMSRVPROCESSSVC procedure acquires the names of the columns that the service expects to receive as input from the DataFlux Data Management service. Since services can be created where any named data set is used for input, no input data set name is required for the DMSRVPROCESSSVC procedure.

The Output Data Set

If the output data set exists, new output data overwrites any existing data. The type of the output data is determined by the service.

Example: Run a DataFlux Data Management Service

Features:

PROC DMSRVPROCESSSVC
  HOST=
  PORT=
  SERVICE=
  TIMEOUT=
  USERID=
  PASSWORD=
RUN;

This example runs a DataFlux Data Management service on a DataFlux Data Management Server that is installed on the default port, Port 21036, of the ‘myserver’ server. In the example, the service is specified and, with the time-out value of 360, the job will terminate after 360 seconds if it does not complete within that time. The job is executed under the credentials (user ID and password) that are specified in the procedure. The SERVICE was previously created and uploaded to the DataFlux Data Management Server.

PROC DMSRVPROCESSSVC
  HOST='http://myhost.unx.com'   PORT=21036
  SERVICE='concatenate.djf'
  TIMEOUT=360
  USERID='myname'
  PASSWORD='mypassword'
RUN;
Chapter 7

DQLOCLST Procedure

Overview: DQLOCLST

What Does the DQLOCLST Procedure Do?

The SAS procedure DQLOCLST generates a list of locales contained in the Quality Knowledge Base (QKB) that is named by the SAS option DQSETUPLOC. The list of locales is written into a SAS data set that the user selects. This data set contains a single column, LOCALE, whose value is a locale name. There is one row per locale found in the QKB.

Syntax: DQLOCLST Procedure

PROC DQLOCLST
<OUT=output-data-set>;
run;

PROC DQLOCLST Statement

The DQLOCLST procedure creates a data set that includes the list of locales in the Quality Knowledge Base that is named by the SAS option DQSETUPLOC.

Syntax

PROC DQLOCLST
<OUT=output-data-set>;
run;

**Action**

**OUT=output-data-set**
identifies the name of the output data set. The procedure follows standard SAS data set naming conventions.

---

**Example: Create a Data Set of Locales**

**Features:**

PROC DQLOCLST
  OUT=

This example creates a SAS data set with the name `loclist` in the work library. The DQSETLOC procedure specifies the QKB to use. For this example, the QKB is in the `/u/saspgs/dynbf/sample` directory. This example prints the result.

```sas
options DQSETUPLOC="/dynbf/sample";
proc dqloclst out = work.loclist;
  run;
proc print data=work.loclist;
  run;
```
Overview: DQMATCH Procedure

What Does the DQMATCH Procedure Do?

PROC DQMATCH creates match codes as a basis for standardization or transformation. The match codes reflect the relative similarity of data values. Match codes are created based on a specified match definition in a specified locale. The match codes are written to an output SAS data set. Values that generate the same match codes are candidates for transformation or standardization.

The DQMATCH procedure can generate cluster numbers for input values that generate identical match codes. Cluster numbers are not assigned to input values that generate unique match codes. Input values that generate a unique match code (no cluster number) can be excluded from the output data set. Blank values can be retained in the output data set, and they can receive a cluster number.

A specified sensitivity level determines the amount of information in the match codes. The amount of information in the match code determines the number of clusters and the number of entries in each cluster. Higher sensitivity–levels produce fewer clusters, with fewer entries per cluster. Use higher sensitivity–levels when you need matches that are more exact. Use lower sensitivity–levels to sort data into general categories or to capture all values that use different spellings to convey the same information.
Syntax: DQMATCH Procedure

At least one CRITERIA statement is required.

Match codes are an encoded version of a character value that is created as a basis for data analysis and data cleansing. Match codes are used to cluster and compare character values. The DQMATCH procedure generates match codes based on a definition and sensitivity value.

**PROC DQMATCH** DATA=**input-data-set-name**

CLUSTER=**output-numeric-variable-name**

CLUSTER_BLANKS | NO_CLUSTER_BLANKS

CLUSTERS_ONLY

DELIMITER | NODELIMITER

LOCALE=locale-name

MATCHCODE=**output-character-variable-name**

OUT=**output-data-set-name**;

**CRITERIA** option(s);

PROC DQMATCH Statement

Create match-codes as a basis for standardization or transformation.

**Syntax**

**PROC DQMATCH** <DATA=**input-data-set-name**>

<CLUSTER=**output-numeric-variable-name**>

<CLUSTER_BLANKS | NOCLUSION_BLANKS>

<CLUSTERS_ONLY>

<DELIMITER | NODELIMITER>

<LOCALE=locale-name>

<MATCHCODE=**output-character-variable-name**>

<OUT=**output-data-set-name**>;

**Optional Arguments**

**CLUSTER=variable-name**

specifies the name of the numeric variable in the output data set that contains the cluster number.

**Interaction**

If the **CLUSTER=** option is not specified and if the **CLUSTERS ONLY option** is specified, an output variable named **CLUSTER** is created.

**CLUSTER BLANKS | NO CLUSTER BLANKS**

specifies how to process blank values.

**CLUSTER BLANKS**

specifies that blank values are written to the output data set. The blank values do not have accompanying match codes.
NO_CLUSTER_BLANKS

specifies that blank values are not written to the output data set.

Default CLUSTER_BLANKS

CLUSTERS_ONLY

specifies that input character values that are part of a cluster are written to the output data set. Excludes input character values that are not part of a cluster.

Default This option is not asserted by default. Typically, all input values are included in the output data set.

Note A cluster number is assigned only when two or more input values produce the same match-code.

DATA= input-data-set-name

specifies the name of the input SAS data set.

Default The most recently created data set in the current SAS session.

DELIMITER | NODELIMITER

specifies whether exclamation points (!) are used as delimiters.

DELIMITER

when multiple CRITERIA statements are specified, DELIMITER specifies that exclamation points (!) separate the individual match codes that make up the concatenated match code. Match codes are concatenated in the order of appearance of CRITERIA statements in the DQMATCH procedure.

NODELIMITER

specifies that multiple match codes are concatenated without exclamation point delimiters.

Defaults (SAS) uses a delimiter.

(DataFlux Data Management Studio) does not use a delimiter.

Note Be sure to use delimiters consistently if you plan to analyze, compare, or combine match codes created in SAS and in DataFlux Data Management Studio.

LOCALE=locale-name

specifies the name of the locale that is used to create match codes. The locale-name can be a name in quotation marks, or an expression that evaluates to a locale-name. It can also be the name of a variable whose value is a locale-name.

The specified locale must be loaded into memory as part of the locale list. If you receive an out-of-memory error when you load the locale, you can increase the value in the MAXMEMQUERY system option. For more information, see your host-specific SAS 9.4 documentation, such as SAS Companion for Windows.

Default The first locale name in the locale list.

Restriction If no locale-name is specified, the first locale in the locale list is used.

Note The match definition, which is part of a locale, is specified in the CRITERIA statement. This specification allows different match definitions to be applied to different variables in the same procedure.
MATCHCODE=**output-character-variable-name**
specifies the name of the output character variable that stores the match codes. The DQMATCH procedure defines a sufficient length for this variable, even if a variable with the same name exists in the input data set.

MATCH_CD is created if the following statements are all true:
- The MATCHCODE= option is not specified in the DQMATCH procedure.
- The MATCHCODE= option is not specified in subsequent CRITERIA statements.
- The CLUSTER= option is not specified.
- The CLUSTERS_ONLY= option is not specified.

Type The MATCHCODE variable always has the data type CHAR.

OUT=**output-data-set-name**
specifies the name of the output data set for match codes created with the DQMATCH procedure. The DQMATCH procedure creates match codes for specified character variables in an input data set.

Note If the specified output data set does not exist, the DQMATCH procedure creates it.

---

**CRITERIA Statement**

Creates match codes and optional cluster numbers for an input variable.

**Syntax**

```
CRITERIA <CONDITION=integer>
  <DELIMSTR=|VAR=|VAR=value-name | VAR=|VAR=value-name>
  <EXACT | MATCHDEF>
  <MATCHCODE=|MATCHCODE=|MATCHCODE=value-name>
  <SENSITIVITY=sensitivity-level >;
```

**Optional Arguments**

**CONDITION=integer**
groups CRITERIA statements to constrain the assignment of cluster numbers.

- Multiple CRITERIA statements with the same CONDITION= value are all required to match the values of an existing cluster to receive the number of that cluster.
- The CRITERIA statements are applied as a logical AND.
- If more than one CONDITION= option is defined in a series of CRITERIA statements, then a logical OR is applied across all CONDITION= option values.
- In a table of customer information, you can assign cluster numbers based on matches between the customer name AND the home address.
- You can also assign cluster numbers on the customer name and organization address.
• All CRITERIA statements that lack a CONDITION= option receive a cluster number based on a logical AND of all such CRITERIA statements.

**Default**  
1

**Restriction**  
If you specify a value for the MATCHCODE= option in the DQMATCH procedure, and you specify more than one CONDITION= value, SAS generates an error. To prevent the error, specify the MATCHCODE= option in CRITERIA statements only.

**Note**  
If you have not assigned a value to the CLUSTER= option in the DQMATCH procedure, cluster numbers are assigned to a variable named CLUSTER by default.

**See**  
“DQMATCHINFOGET Function” on page 105.

**DELIMSTR** | **VAR**  
specifies the name of a variable.

**DELIMSTR=input-variable-name**  
specifies the name of a variable that has been parsed by the DQPARSE function, or contains tokens added with the DQPARSETOKENPUT function.

**VAR=input-variable-name**  
specifies the name of the character variable that is used to create match codes. If the variable contains delimited values, use the DELIMSTR= option.

**Restrictions**  
The values of this variable cannot contain delimiter added with the DQPARSE function or the DQPARSETOKENPUT function.

You cannot specify the DELIMSTR= option and the VAR= option in the same CRITERIA statement.

**See**  
“DQPARSE Function” on page 108 for additional information.

“DQPARSETOKENPUT Function” on page 114 for additional information.

**EXACT** | **MATCHDEF**  
assigns a cluster number.

**EXACT**  
assigns a cluster number based on an exact character match between values.

**Restriction**  
If you specify the EXACT= option, you cannot specify the MATCHDEF= option, the MATCHCODE= option, or the SENSITIVITY= option.

**MATCHDEF=match-definition**  
specifies the match-definition that is used to create the match code for the specified variable.

**Restrictions**  
The match-definition must exist in the locale that is specified in the LOCALE= option of the DQMATCH procedure.

If you specify the MATCHDEF= option, you cannot specify the EXACT option, the MATCHCODE= option, or the SENSITIVITY option.
If the CLUSTER= option has not been assigned a variable in the DQMATCH procedure, then cluster numbers are assigned to the variable named CLUSTER.

Restriction: If you specify the MATCHCODE= option in the DQMATCH procedure, the match–code is a composite of the exact character-value and the match code that is generated by the match-definition.

MATCHCODE=character-variable
specifies the name of the variable that receives the match codes for the character variable that is specified in the VAR= option or the DELIMSTR= option.

Type: The MATCHCODE variable always has the data type CHAR.

Restrictions: The MATCHCODE= option in the CRITERIA statement is not valid if you also specify the MATCHCODE= option in the DQMATCH procedure. If you are using multiple CRITERIA statements, either specify the MATCHCODE= option in each CRITERIA statement or specify the option only in the DQMATCH procedure. Specifying the MATCHCODE= option in the DQMATCH procedure generates composite match codes.

SENSITIVITY=sensitivity-level
determines the amount of information in the resulting match codes. Higher sensitivity values create match codes that contain more information about the input values. Higher sensitivity levels result in a greater number of clusters, with fewer values in each cluster.

Default: The default value is 85.

Details
Match codes are created for the input variables that are specified in each CRITERIA statement. The resulting match codes are stored in the output variables that are named in the MATCHCODE= option. The MATCHCODE= option can be specified in the DQMATCH procedure or the CRITERIA statement.

Simple match codes are created when the CRITERIA statements specify different values for their respective MATCHCODE= options. Composite match codes are created when two or more CRITERIA statements specify the same value for their respective MATCHCODE= options.

To create match codes for a parsed character variable, specify the DELIMSTR= option instead of the VAR= option. In the MATCHDEF= option, be sure to specify the name of the match-definition. This definition is associated with the parse definition that was used to add delimiters to the character variable. To determine the parse definition that is associated with a match definition, use the DQMATCHINFOGET function.
Examples: DQMATCH Procedure

Example 1: Generate Composite Match Codes

The following example uses the DQMATCH procedure to create composite match codes and cluster numbers. The default sensitivity level of 85 is used in both CRITERIA statements. The locale ENUSA is assumed to have been loaded into memory previously with the %DQLOAD AUTOCALL macro.

/* Create the input data set. */
data cust_db;
  length customer $ 22;
  length address $ 31;
  input customer $char22. address $char31.;
datalines;
Bob Beckett 392 S. Main St. PO Box 2270
Robert E. Beckett 392 S. Main St. PO Box 2270
Rob Beckett 392 S. Main St. PO Box 2270
Paul Becker 392 N. Main St. PO Box 7720
Bobby Becket 392 Main St.
Mr. Robert J. Beckeit P. O. Box 2270 392 S. Main St.
Mr. Robert E Beckett 392 South Main Street #2270
Mr. Raul Becker 392 North Main St.
;
run;

/* Run the DQMATCH procedure. */
proc dqmatch data=cust_db out=out_db1 matchcode=match_cd
  cluster=clustergrp locale='ENUSA';
criteria matchdef='Name' var=customer;
criteria matchdef='Address' var=address;
run;

/* Print the results. */
proc print data=out_db1;
run;
Output 8.1 PROC Print Output

Details
The output data set, OUT_DB1, includes the new variables MATCH_CD and CLUSTERGRP. The MATCH_CD variable contains the composite match code that represents both the customer name and address. Because the default argument DELIMITER was used, the resulting match code contains two match code components (one from each CRITERIA statement) that are separated by an exclamation point.

The CLUSTERGRP variable contains values that indicate that five of the character values are grouped in a single cluster and that the other three are not part of a cluster. The clustering is based on the values of the MATCH_CD variable. By looking at the values for MATCH_CD, you can see that five character values have identical match code values. Although the match code value for customer Bobby Becket is similar to the Cluster 1 match codes, the address difference caused it to be excluded in Cluster 1.

“Example 2: Matching Values Using Mixed Sensitivity Levels” on page 44 shows how the use of non-default sensitivity levels increases the accuracy of the analysis.

Note: This example is available in the SAS Sample Library under the name DQMCDFLT.

Example 2: Matching Values Using Mixed Sensitivity Levels
The following example is similar to “Example 1: Generate Composite Match Codes” on page 43 in that it displays match codes and clusters for a simple data set. This example differs in that the CRITERIA statement for the ADDRESS variable uses a sensitivity of 50. The CRITERIA statement for the NAME variable uses the same default sensitivity of 85.

The use of mixed sensitivities enables you to customize your clusters for maximum accuracy. In this case, clustering accuracy is increased when the sensitivity level of a less important variable is decreased.

This example primarily shows how to identify possible duplicate customers based on their names. To minimize false duplicates, minimal sensitivity is applied to the addresses.

/* Create the input data set. */
data cust_db;
length customer $ 22;
length address $ 31;
input customer $char22. address $char31.;
datalines;
Bob Beckett             392 S. Main St. PO Box 2270
Robert E. Beckett       392 S. Main St. PO Box 2270
Rob Beckett             392 S. Main St. PO Box 2270
Paul Becker             392 N. Main St. PO Box 7720
Bobby Becket            392 Main St.
Mr. Robert J. Beckeit   P. O. Box 2270 392 S. Main St.
Mr. Robert E Beckett    392 South Main Street #2270
Mr. Raul Becker         392 North Main St.
;
run;

/* Run the DQMATCH procedure. */
proc dqmatch data=cust_db out=out_db2 matchcode=match_cd
cluster=clustergrp locales='ENUSA';
criteria matchdef='Name' var=customer;
criteria matchdef='Address' var=address sensitivity=50;
run;

/* Print the results. */
proc print data=out_db2;
run;

Output 8.2 PROC Print Output

Details
The output data set, OUT_DB2, includes the new variables MATCH_CD and CLUSTERGRP. The MATCH_CD variable contains the match code that represents both the customer name and address. Because the default argument DELIMITER was used, the resulting match code contains two match code components (one from each CRITERIA statement) that are separated by an exclamation point.

The CLUSTERGRP variable contains values that indicate that six of the character values are grouped in a single cluster and that the other two are not part of any cluster. The clustering is based on the values of the MATCH_CD variable.
This result is different than in “Example 1: Generate Composite Match Codes” on page 43, where only five values were clustered based on NAME and ADDRESS. This difference is caused by the lower sensitivity setting for the ADDRESS criteria in the current example. This makes the matching less sensitive to variations in the address field. Therefore, the value Bobby Becket has now been included in Cluster 1.392 Main St. is considered a match with 392 S. Main St. PO Box 2270 and the other variations, this was not true at a sensitivity of 85.

Note: This example is available in the SAS Sample Library under the name DQMCMIXD.

Example 3: Matching Values Using Minimal Sensitivity

The following example shows how minimal sensitivity levels can generate inaccurate clusters. A sensitivity of 50 is used in both CRITERIA statements, which is the minimum value for this argument.

```sas
/* Create the input data set. */
data cust_db;
  length customer $ 22;
  length address $ 31;
  input customer $char22. address $char31.;
datalines;
Bob Beckett             392 S. Main St. PO Box 2270
Robert E. Beckett       392 S. Main St. PO Box 2270
Rob Beckett             392 S. Main St. PO Box 2270
Paul Becker             392 N. Main St. PO Box 7720
Bobby Becket            392 Main St.
Mr. Robert J. Beckett   P. O. Box 2270 392 S. Main St.
Mr. Robert E Beckett    392 South Main Street #2270
Mr. Raul Becker         392 North Main St.
;
run;

/* Run the DQMATCH procedure. */
proc dqmatch data=cust_db out=out_db3 matchcode=match_cd
cluster=clustergrp locale='ENUSA';
criteria matchdef='Name' var=customer sensitivity=50;
criteria matchdef='Address' var=address sensitivity=50;
run;

/* Print the results. */
proc print data=out_db3;
run;
```
The output data set OUT_DB3 includes the variables MATCH_CD and CLUSTERGRP. The MATCH_CD variable contains the match code that represents both the customer name and address. Because the default argument DELIMITER was used, the resulting match code contains two match code components (one from each CRITERIA statement) that are separated by an exclamation point.

The CLUSTERGRP variable contains values that indicate that six of the values are grouped in one cluster and that the other two are grouped in another. The clustering is based on the values of the MATCH_CD variable. This example shows that, with a minimal sensitivity level of 50, the following values match and form a cluster.

Mr. Raul Beckett
Paul Becker

A higher sensitivity level would not cluster these observations.

Note: This example is available in the SAS Sample Library under the name DQMCMIN.

---

**Example 4: Creating Match Codes for Parsed Values**

The following example creates match codes for parsed character data. The program loads locales, determines a parse definition, creates character elements, creates parsed character values, and creates match codes for the parse character elements.

This example is available in the SAS Sample Library under the name DQMCPARS.

```sas
/* load locales */
%dqload(dqlocale=(enusa),
dqsetuploc=('your-dqsetup-file-here')

/* Determine the parse definition associated with your */
/* match definition. */
data _null_;
parsedefn=dqMatchInfoGet('Name');
call symput('parsedefn', parsedefn);
p = 'The parse definition for the NAME match definition is: ' parsedefn;
tokens=dqParseInfoGet(parsedefn);
p = 'The ' parsedefn 'parse definition tokens are: ' / @5 tokens;
run;
```
/* Create variables containing name elements. */
data parsed;
  length first last $ 20;
  first='Scott'; last='James';  output;
  first='James'; last='Scott';  output;
  first='Ernie'; last='Hunt';  output;
  first='Brady'; last='Baker';  output;
  first='Ben'; last='Riedel';  output;
  first='Sara'; last='Fowler';  output;
  first='Homer'; last='Webb';  output;
  first='Poe'; last='Smith';  output;
run;

/* Create parsed character values. */
data parsedview;
  set parsed;
  length delimstr $ 100;

  * Insert one token at a time;
  delimstr=dqParseTokenPut(delimstr, first, 'Given Name', 'Name');
  delimstr=dqParseTokenPut(delimstr, last,  'Family Name', 'Name');
run;

/* Generate match codes using the parsed character values. */
proc dqmatch data=parsedview
  out=mcodes;
  criteria matchdef='Name' delimstr=delimstr sensitivity=85;
run;

/* Print the match codes. */
proc print data=mcodes;
  title 'Look at the match codes from PROC DQMATCH';
run;

Output 8.4  PROC Print Output

<table>
<thead>
<tr>
<th>Obs</th>
<th>first</th>
<th>last</th>
<th>delimstr</th>
<th>MATCH_CD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Scott</td>
<td>James</td>
<td>/Scott://James://</td>
<td>B94854452J&lt;&lt;&lt;&lt;</td>
</tr>
<tr>
<td>2</td>
<td>James</td>
<td>Scott</td>
<td>/James://Scott://</td>
<td>4511111C7E&lt;&lt;&lt;&lt;</td>
</tr>
<tr>
<td>3</td>
<td>Ernie</td>
<td>Hunt</td>
<td>/Ernie://Hunt://</td>
<td>2P-6886_YF&lt;&lt;&lt;&lt;</td>
</tr>
<tr>
<td>4</td>
<td>Brady</td>
<td>Baker</td>
<td>/Brady://Baker://</td>
<td>K9,YSSS88MY-&lt;&lt;SSSS</td>
</tr>
<tr>
<td>5</td>
<td>Ben</td>
<td>Riedel</td>
<td>/Ben://Riedel://</td>
<td>9V8Y8881Y,F&lt;&lt;&lt;&lt;</td>
</tr>
<tr>
<td>6</td>
<td>Sara</td>
<td>Fowler</td>
<td>/Sara://Fowler://</td>
<td>9WY8888BY&lt;&lt;&lt;&lt;</td>
</tr>
<tr>
<td>7</td>
<td>Homer</td>
<td>Webb</td>
<td>/Homer://Webb://</td>
<td>LM&lt;&lt;&lt;&lt;S2BY&lt;&lt;&lt;&lt;</td>
</tr>
<tr>
<td>8</td>
<td>Poe</td>
<td>Smith</td>
<td>/Poe://Smith://</td>
<td>4B-25554V&lt;&lt;&lt;&lt;</td>
</tr>
</tbody>
</table>
Example 5: Clustering with Multiple CRITERIA Statements

The following example assigns cluster numbers based on a logical OR of two pairs of CRITERIA statements. Each pair of CRITERIA statements is evaluated as a logical AND. The cluster numbers are assigned based on a match between the customer name and address, or the organization name and address.

/* Load the ENUSA locale. The system option DQSETUPLOC= is already set. */

%dqload(dqlocale=(enusa))

data customer;
  length custid 8 name org addr $ 20;
  input custid name $char20. org $char20. addr $char20.;
datalines;
  1  Mr. Robert Smith    Orion Star Corporation   8001 Weston Blvd.
  2                      The Orion Star Corp.     8001 Westin Ave
  3  Bob Smith                                    8001 Weston Parkway
  4  Sandi Booth         Belleview Software       123 N Main Street
  5  Mrs. Sandra Booth   Belleview Inc.           801 Oak Ave.
  6  sandie smith Booth  Orion Star Corp.         123 Maine Street
  7  Bobby J. Smythe     ABC Plumbing             8001 Weston Pkwy
;
run;

/* Generate the cluster data. Because more than one condition is defined, a variable named CLUSTER is created automatically */

proc dqmatch data=customer
  out=customer_out;
  criteria condition=1 var=name sensitivity=85 matchdef='Name';
  criteria condition=1 var=addr sensitivity=70 matchdef='Address';
  criteria condition=2 var=org sensitivity=85 matchdef='Organization';
  criteria condition=2 var=addr sensitivity=70 matchdef='Address';
run;

/* Print the result. */

proc print data=customer_out noobs;
run;
Details
In the preceding output, the two rows in cluster 1 matched on name and address. The rows in cluster 2 matched on name and address as well as organization and address. The inclusion of Bobby J. Smythe in cluster 2 indicates either a data error or a need for further refinement of the criteria and conditions. The last row in the output did not receive a cluster number because that row did not match any other rows.

Note: This example is available in the SAS Sample Library under the name DQMLTCND.

Example 6: Generating Multiple Simple Match Codes
The following example creates more than one simple match code with a single DQMATCH procedure step. The first example, created a composite match code by specifying the MATCHCODE= option in the DQMATCH procedure statement.

This example creates simple match codes by specifying the MATCHCODE= option in each CRITERIA statement. In addition, unlike the first example, which creates a cluster number, you cannot create a cluster number when generating multiple simple match codes.

The default sensitivity level of 85 is used in both CRITERIA statements. The locale ENUSA is assumed to have been loaded into memory previously with the %DQLOAD AUTOCALL macro.

/* Create the input data set. */
data cust_db;
  length customer $ 22;
  length address $ 31;
  input customer $char22. address $char31.;
datalines;
Bob Beckett             392 S. Main St. PO Box 2270
Robert E. Beckett       392 S. Main St. PO Box 2270
Rob Beckett             392 S. Main St. PO Box 2270
Paul Becker             392 N. Main St. PO Box 7720
Bobby Becket            392 Main St.
Example 6: Generating Multiple Simple Match Codes

Mr. Robert J. Beckeit     P. O. Box 2270 392 S. Main St.
Mr. Robert E Beckett      392 South Main Street #2270
Mr. Raul Becker           392 North Main St.

/* Run the DQMATCH procedure. */
proc dqmatch data=cust_db out=out_db5 locale='ENUSA';
  criteria matchdef='Name' var=customer matchcode=mc_name;
  criteria matchdef='Address' var=address matchcode=mc_addr;
run;

/* Print the results. */
proc print data=out_db5;
run;

Output 8.6  PROC Print Output

<table>
<thead>
<tr>
<th>Obs</th>
<th>customer</th>
<th>address</th>
<th>MC_NAME</th>
<th>MC_ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bob Beckett</td>
<td>392 S. Main St. P.O Box 2270</td>
<td>M000000000000</td>
<td>K000000000000</td>
</tr>
<tr>
<td>2</td>
<td>Bob Beckett</td>
<td>392 S. Main St. P.O Box 2270</td>
<td>M000000000000</td>
<td>K000000000000</td>
</tr>
<tr>
<td>3</td>
<td>Bob Beckett</td>
<td>392 S. Main St. P.O Box 7720</td>
<td>M000000000000</td>
<td>K000000000000</td>
</tr>
<tr>
<td>4</td>
<td>Bob Beckett</td>
<td>392 N. Main St. P.O Box 7720</td>
<td>M000000000000</td>
<td>K000000000000</td>
</tr>
<tr>
<td>5</td>
<td>Bob Beckett</td>
<td>392 N. Main St. P.O Box 2270</td>
<td>M000000000000</td>
<td>K000000000000</td>
</tr>
<tr>
<td>6</td>
<td>Mr. Robert J. Beckeit</td>
<td>P. O. Box 2270 392 S. Main St.</td>
<td>M000000000000</td>
<td>K000000000000</td>
</tr>
<tr>
<td>7</td>
<td>Mr. Robert E. Beckett</td>
<td>392 South Main Street #2270</td>
<td>M000000000000</td>
<td>K000000000000</td>
</tr>
<tr>
<td>8</td>
<td>Mr. Raul Becker</td>
<td>392 North Main St.</td>
<td>M000000000000</td>
<td>K000000000000</td>
</tr>
</tbody>
</table>

Details

The output data set, OUT_DB5, includes the new variables MC_NAME and MC_ADDR. Compare this to the result in example 1, where the same match code values were combined to form a composite match code in the MATCH_CD variable.

Using simple or composite match codes depends on the type of comparison that you need. If you want to compare names and addresses separately, generate separate match codes as shown in this example. If you want to do comparisons based on the combined Name and Address, generate a composite match code as shown in example 1.

See “Example 1: Generate Composite Match Codes” on page 43 to compare the examples.

Note: This example is available in the SAS Sample Library under the name DQMCDFL2.
Overview: DQSCHEME Procedure

What Does the DQSCHEME Procedure Do?

PROC DQSCHEME creates scheme data sets and analysis data sets and applies schemes to input data sets. You can also apply schemes with the DQSCHEMEAPPLY function or CALL routine. See “DQSCHEMEAPPLY CALL Routine” on page 120.

The DQSCHEME procedure enables you to create and apply schemes that transform similar data values into the single most common value, as shown in the following diagram.
The DQSCHEME procedure also analyzes and reports on the quality of your data.

**Syntax: DQSCHEME Procedure**

```
PROC DQSCHEME DATA=input-data-set
   <QKB | NOQKB>
   OUT=output-data-set;
   APPLY <option(s)>;
   CONVERT <option(s)>;
   CREATE <option(s)>;
```

**PROC DQSCHEME Statement**

Creates scheme data sets and analysis data sets and applies schemes to input data sets.

**Syntax**

```
PROC DQSCHEME <DATA=input-data-set>
   <QKB | NOQKB>
   <OUT=output-data-set>;
```

**Optional Arguments**

**QKB | NOQKB**

specifies whether the schemes will be in QKB scheme file format or SAS format.

**QKB**

specifying QKB indicates that all schemes are in QKB scheme file format.

**NOQKB**

specifying NOQKB indicates that all schemes are in SAS format. The DQSCHEME procedure can create and apply schemes in either format. Schemes in QKB scheme file format can be edited using the feature-rich graphical user interface of the DataFlux Data Management Studio software.

Default: QKB
Restrictions

Always specify NOQKB when creating schemes in the z/OS operating environment.

In schemes stored in SAS format, data set labels are used to store meta options. Therefore, you should not specify data set labels in scheme data sets that are stored in SAS format. If you specify data set labels, you overwrite the scheme metadata.

See

“Meta Options” on page 61.

DATA= input-data-set

When you use the CREATE statement to create schemes, the DATA= option specifies the SAS data set from which one or more schemes are built. When you use the APPLY statement to apply existing schemes, the DATA= option specifies the SAS data set against which schemes are applied.

Default

The most recently created data set in the current SAS session.

OUT= output-data-set

specifies the output data set.

Interactions

If the specified data set does not exist, the DQSCHME procedure creates it.

If you use one or more APPLY statements, you must use the OUT= option to specify the name of the output data set.

If you specify OUT= without any APPLY statements, an empty output data set is created.

Results are written to the output data set after all schemes have been applied.

APPLY Statement

Applies a scheme to transform the values of a single variable.

Syntax

APPLY <LOCALE= locale-name>

<MATCH-DEFINITION=match-definition>

<MODE=ELEMENT | PHRASE>

<SCHHEME=scheme-name>

<SCHHEME_LOOKUP =EXACT | IGNORE_CASE | USE_MATCHDEF>

SENSITIVITY=sensitivity-level>

<VAR=variable-name>;

Optional Arguments

LOCALE=locale-name

specifies the name of the match definition, in the specified locales, that is used to create match codes that run the application of the scheme.
MATCH-DEFINITION=

specifies the name of the match definition, in the specified locales, that is used to create match codes that run the application of the scheme.

MODE=ELEMENT | PHRASE

specifies a mode of scheme application. This information is stored in the scheme as metadata, which specifies a default mode when the scheme is applied. The default mode is superseded by a mode in the APPLY statement, or in the DQSCHEMEAPPLY function or CALL routine.

ELEMENT

specifies that each element in each value of the input character variable is compared to the data values in the scheme. When SCHEME_LOOKUP = USE_MATCHDEF, the match code for each element is compared to match codes generated for each element, in each DATA variable value in the scheme.

PHRASE

this default value specifies that the entirety of each value of the input character variable is compared to the data values in the scheme. When SCHEME_LOOKUP = USE_MATCHDEF, the match code for the entire input value is compared to match codes that are generated for each data value in the scheme.

SCHEME= scheme-name

identifies the scheme to apply to the input data set. In all the operating environments other than z/OS, schemes using QKB scheme file format are identified by specifying a fileref or a fully qualified name that ends in .sch.qkb.

SCHEME_LOOKUP = EXACT | IGNORE_CASE | USE_MATCHDEF

specifies the method of applying the scheme to the data values of the input variable.

SENSITIVITY=sensitivity-level

specifies the amount of information in the resulting match codes.

VAR=variable-name

specifies the name of the variable or source column that is analyzed and transformed.

Note: A variable in a CAS table with the data type VARCHAR will be written to output as data type VARCHAR if the output table is a CAS table. Otherwise, the variable will be written to output as data type CHAR. For more information, see SAS Cloud Analytic Services: Language Reference and SAS Cloud Analytic Services: DATA Step Programming.

CONVERT Statement

Converts schemes between SAS and QKB scheme file formats.

Requirement: All options are required.

Syntax

CONVERT <QKBTOSAS | SASTOQKB>

<IN=input-data-set>

<OUT=output-data-set>;
**Required Arguments**

**QKBTOSAS | SASTOQKB**

Specify QKBTOSAS to convert a scheme in QKB scheme file format to SAS format. Specify SASTOQKB to convert a scheme in SAS format to QKB scheme file format. Schemes in SAS format are created with the CREATE statement using the NOQKB option in the DQSCHME procedure.

**CAUTION** In the z/OS operating environment, specify QKBTOSAS only. In z/OS, schemes in QKB scheme file format can be applied but not created.

**IN=**scheme-data-set

identifies the existing scheme data set that is to be converted.

If QKBTOSAS is specified, then the value must be the name of a fileref that references a fully qualified path in lowercase that ends in **.sch.qkb**.

If SASTOQKB is specified, then the value must be a one-level or two-level SAS data set name.

**Note** In the z/OS operating environment, the PDS specification has no special naming requirements.

**OUT=**converted-scheme-data-set

specifies the name of the data set with the converted scheme.

**Requirements**

If SASTOQKB is specified, the value must be the name of a fileref. This fileref references a fully qualified path in lowercase that ends in **.sch.qkb**.

If QKBTOSAS is specified, the value must be a one-level or two-level SAS data set name.

**Note** The z/OS operating environment, the PDS specification has no special naming requirements.

---

**CREATE Statement**

Creates a scheme or an analysis data set.

**Syntax**

```plaintext
CREATE = <ANALYSIS=analysis-data-set >
  <INCLUDE_ALL>
  <LOCAL=locale-name>
  <MATCHDEF=match-definition >
  <MODE=PHRASE | ELEMENT>
  <SCHEME=scheme-name>
  <SCHEME_LOOKUP=EXACT | IGNORE_CASE | USE_MATCHDEF>
  <SENSITIVITY=sensitivity-level>
  <VAR=input-character-variable> ;
```
**Optional Arguments**

**ANALYSIS=** *analysis-data-set*

Names the output data set that stores analytical data.

*Restriction*  This option is required if the **SCHEME=** option is not specified.

**INCLUDE_ALL**

specifies that the scheme is to contain all of the values of the input variable. This includes input variables with these conditions:

- with unique match codes
- that were not transformed
- that did not receive a cluster number

*Note*  The **INCLUDE_ALL** option is not set by default.

**LOCALE=** *locale-name*

specifies the locale that contains the specified match definition. The value can be a locale name in quotation marks. It can be the name of a variable whose value is a locale name, or is an expression that evaluates to a locale name.

The specified locale must be loaded into memory as part of the locale list.

*Default*  The first locale in the locale list.

*Restriction*  If no value is specified, the default locale is used.

**MATCHDEF=** *match-definition*

names the match definition in the specified locale that is used to establish cluster numbers. You can specify any valid match definition.

The value of the **MATCHDEF=** option is stored in the scheme as a meta option. This provides a default match definition when a scheme is applied. This meta option is used only when **SCHEME_LOOKUP=** MATCHDEF. The default value that is supplied by this meta option is superseded by match definitions specified in the APPLY statement or the DQSCHEMEAPPLY CALL routine.

*Tip*  Use definitions whose names end in **(SCHEME BUILD)** when using the ENUSA locale. These match definitions yield optimal results in the DQSCHEME procedure.

*See*  “Meta Options” on page 61 for additional information.

**MODE=** **ELEMENT** | **PHRASE**

specifies a mode of scheme application. This information is stored in the scheme as metadata, which specifies a default mode when the scheme is applied. The default mode is superseded by a mode in the APPLY statement, or in the DQSCHEMEAPPLY function or CALL routine.

**ELEMENT**

specifies that each element in each value of the input character variable is compared to the data values in the scheme. When **SCHEME_LOOKUP=** USE_MATCHDEF, the match code for each element is compared to match codes generated for each element in each DATA variable value in the scheme.

**PHRASE**

(default value) specifies that the entirety of each value of the input character variable is compared to the data values in the scheme. When
SCHEME_LOOKUP= USE_MATCHDEF, the match code for the entire input value is compared to match codes that are generated for each data value in the scheme.

SCHEME= scheme-name

specifies the name or the fileref of the scheme that is created. The fileref must reference a fully qualified path with a filename that ends in .sch.qkb. Lowercase letters are required. To create a scheme data set in QKB scheme file format, specify the QKB option in the DQSCHHEME procedure.

To create a scheme in SAS format, specify the NOQKB option in the DQSCHHEME procedure and specify a one-level or two-level SAS data set name.

Restriction

The SCHEME= option is required if the ANALYSIS= option is not specified.

See

“Syntax” on page 54 for additional information.

CAUTION

In the z/OS operating environment, specify only schemes that use SAS formats. QKB schemes can be applied, but not created in the z/OS operating environment.

SCHEME_LOOKUP= EXACT | IGNORE_CASE | USE_MATCHDEF

specifies one of three mutually exclusive methods of applying the scheme to the values of the input character variable. Valid values are defined as follows:

EXACT

(default value) specifies that the values of the input variable are to be compared to the DATA values in the scheme without changing the input values in any way. The transformation value in the scheme is written into the output data set only when an input value exactly matches a DATA value in the scheme. Any adjacent blank spaces in the input values are replaced with single blank spaces before comparison.

IGNORE_CASE

specifies that capitalization is to be ignored when input values are compared to the DATA values in the scheme.

Interaction

Any adjacent blank spaces in the input values are replaced with single blank spaces before comparison.

USE_MATCHDEF

specifies that comparisons are to be made between the match codes of the input values and the match codes of the DATA values in the scheme.

Interactions

Specifying USE_MATCHDEF enables the options LOCALE=, MATCHDEF=, and SENSITIVITY=, which can be used to override the default values that might be stored in the scheme.

A transformation occurs when the match code of an input value is identical to the match code of a DATA value in the scheme.

The value of the SCHEME_LOOKUP= option is stored in the scheme as a meta option. This specifies a default lookup method when the scheme is applied. The default supplied by this meta option is superseded by a lookup method that is specified in the APPLY statement, or in the DQSCHHEME_APPLY function or CALL routine.

See

“Meta Options” on page 61 for additional information.
SENSITIVITY=sensitivity-level
determines the amount of information that is included in the match codes that are
generated during the creation and perhaps the application of the scheme. The value
of the SENSITIVITY= option is stored in the scheme as a meta option. This provides
a default sensitivity value when the scheme is applied.

Higher sensitivity values generate match codes that contain more information. These
match codes generally result in the following:

- fewer matches
- greater number of clusters
- fewer values in each cluster

Default 85

Interactions The default value supplied by this meta option is superseded by a
sensitivity value specified in the APPLY statement, or in the
DQSCHEMEAPPLY CALL routine.

This meta option is used at apply time only when
SCHEME_LOOKUP= MATCHDEF.

See “Meta Options” on page 61 for additional information.

VAR=input-character-variable
specifies the input character variable that is analyzed and transformed. The
maximum length of input values is 1024 bytes.

Note: A variable in a CAS table with the data type VARCHAR will be written to
output as data type VARCHAR if the output table is a CAS table. Otherwise, the
variable will be written to output as data type CHAR. For more information, see
SAS Cloud Analytic Services: Language Reference and SAS Cloud Analytic
Services: DATA Step Programming.

Using PROC DQSCHEME

Creating Schemes

Schemes are data sets that are first created from a source table, and then later applied to
that source table to standardize the values in that table.

Scheme data sets are created with the CREATE statement. The CREATE statement uses
matching codes to define groups of similar data values. The values that occur most
frequently in each group become the survivor values for each group. When the scheme is
applied, the survivor values replace the other values in the group.

Note: An error message is generated if the length of an input value exceeds 1024 bytes.

Scheme data sets are created in SAS format or in QKB scheme file format. QKB scheme
file format is recognized by SAS and by the DataFlux Data Management software.

The differences between schemes in SAS format and QKB format are defined as
follows:
• The SAS Data Quality Server software can create and apply schemes in SAS format or QKB format.

• Schemes in SAS format and QKB format can be displayed with the SAS table viewer.

• DataFlux Data Management Studio software can create, apply, and edit schemes in QKB format.

• In the z/OS operating environment, the SAS Data Quality Server software can create, apply, and display schemes in SAS format. Schemes in QKB file format can be applied.

To convert scheme data sets from one format to the other, use the CONVERT statement in PROC DQSCHME.

Generating Analysis Data Sets

Analysis data sets show the groupings of like data values in the scheme-building process. These are the groupings from which the standard value is selected. The data sets are generated by specifying the ANALYSIS= option in the CREATE statement of the DQSCHME procedure. The analysis data sets enable you to experiment with different options to create a scheme that provides optimal data cleansing.

The key to optimizing a scheme is to choose a sensitivity value that is most suitable for your data and your goal. You can create a series of analysis data sets using different sensitivity values to compare the results. Changing the sensitivity value changes the clustering of input values.

When you decide on a sensitivity level, you can create the scheme data set by replacing the ANALYSIS= option with the SCHEME= option in the CREATE statement.

The analysis data set contains one observation for each unique input value. Any adjacent blank spaces are removed from the input values. The COUNT variable describes the number of occurrences of that value.

The CLUSTER variable represents the groupings of data values that are similar based on the selected sensitivity. One standard value is selected from each cluster, based on the value with the highest COUNT (frequency).

Specify the INCLUDE_ALL option in the CREATE statement to include all input values in the scheme. This includes the unique input values that did not receive a cluster number in the analysis data set.

See “Creating Schemes” on page 60 for additional information.

Meta Options

Meta options are stored in the scheme when the scheme is created. The options provide default values for certain options of the DQSCHME procedure's APPLY statement. The meta options also store default arguments for the DQSCHMEAPPLY function or CALL routine. Default values are stored for the lookup mode (SCHEME_LOOKUP option or argument), apply mode (MODE option or argument), match definition, and sensitivity level. The values of the meta options are superseded when other values are specified in the APPLY statement or in the DQSCHMEAPPLY function or CALL routine.

Meta options for the match definition and sensitivity value are valid only when the scheme is applied with match code lookup and when SCHEME_LOOKUP=USE_MATCHDEF.
The meta options are stored differently depending on the scheme format. For schemes in SAS format, the meta options are stored in the data set label. For schemes in QKB scheme file format, the meta options are stored within the scheme itself.

*Note:* In programs that create schemes in SAS format, do not specify a data set label; doing so deletes the meta options.

The meta options are stored using:

'lookup-method' 'apply-mode' 'sensitivity-level' 'match-definition'

**lookup-method**
- EM specifies that the default value of the SCHEME_LOOKUP option or argument is EXACT. In order for an input value to be transformed, that value must exactly match a DATA value in the scheme.
- IC specifies that SCHEME_LOOKUP=IGNORE_CASE.
- UM specifies that SCHEME_LOOKUP=USE_MATCHDEF. Match codes are created and compared for all input values and all DATA values in the scheme.

**apply-mode**
- E specifies that the default value of the MODE option or argument is ELEMENT.
- P specifies that MODE=PHRASE.

**sensitivity-level**
- the amount of information in the match codes that is generated when SCHEME_LOOKUP=USE_MATCHDEF.
- Valid values range from 50 to 95.

**match-definition**
- the name of the default match definition that is used when the value of the SCHEME_LOOKUP option is USE_MATCHDEF.

For example, the meta options string, 'UM' 'P' '80' 'NAME', specifies that the scheme:
- lookup method is match code
- the apply-mode is by phrase
- the sensitivity-level is 80
- the match-definition is NAME

---

**Examples: DQSCHEME Procedure**

**Example 1: Creating an Analysis Data Set**

**Overview**
This example generates an analysis of the STATE variable in the VENDORS data set.

*Note:* You do not have to create a scheme to generate the analysis data set.

*Note:* The locale ENUSA is assumed to have been loaded into memory as part of the locale list.
For each value of the STATE variable, the analysis data set WORK.A_STATE shows the number of occurrences and the associated cluster number. Variables that are not clustered with any other values have a blank value for the cluster number.

Note: This example is available in the SAS Sample Library under the name DQANALYZ.

```sas
/* Create the input data set. */
data vendors;
  input city $char16. state $char22. company $char34.;
datalines;
Detroit           MI                      Ford Motor
Dallas            Texas                   Wal-mart Inc.
Washington        District of Columbia  Federal Reserve Bank
SanJose           CA                      Wal mart
New York          New York                Ernst & Young
Virginia Bch       VA                      TRW INC - Space Defense
Dallas            TX                      Walmart Corp.
San Francisco     California              The Jackson Data Corp.
New York          NY                      Ernst & Young
Washington        DC                      Federal Reserve Bank 12th District
New York          N.Y.                    Ernst & Young
San Francisco     CA                      Jackson Data Corporation
Atlanta           GA                      Farmers Insurance Group
RTP               NC                      Kaiser Permanente
New York          NY                      Ernest and Young
Virginia Beach     VIRGINIA                TRW Space & Defense
Detroit           Michigan                Ford Motor Company
San Jose          CA                      Jackson Data Corp
Washington        District of Columbia  Federal Reserve Bank
Atlanta           GEORGIA                 Target
;
run;

/* Create the analysis data set. */
proc dgscheme data=vendors;
  create analysis=a_state
    matchdef='State (Scheme Build)'
    var=state
    locale='ENUSA';
run;

/* Print the analysis data set. */
title 'Analysis of state name variations';
proc print data=a_state;
run;
```
Example 2: Creating Schemes

The following example generates three schemes in SAS format. The match definition for Organization is assumed to be in the QKB used for this code. Note that the locale ENUSA is assumed to have been loaded into memory as part of the locale list.

```sas
/* Create the input data set. */
data vendors;
  input city $char17. state $char22. company $char36.;
datalines;
  Detroit          MI                          Ford Motor
  Dallas           Texas                       Wal-mart Inc.
  Washington       District of Columbia        Federal Reserve Bank
  Washington District of Columbia   Federal Reserve Bank
  Atlanta          GEORGIA                     Target
; run;

proc dqscheme data=vendors nogkb;
  create matchdef='City (Scheme Build)' var=city
    scheme=city_scheme locale='ENUSA';
  create matchdef='State (Scheme Build)' var=state
```
scheme=state_scheme locale='ENUSA';
create matchdef='Organization'
  var=company scheme=org_scheme locale='ENUSA';
run;

title 'City scheme';
proc print data=work.city_scheme;
run;

title 'State scheme';
proc print data=work.state_scheme;
run;

title 'Organization scheme';
proc print data=work.org_scheme;
run;

Details

Notice that this example did not create and immediately apply one or more schemes within the same step. After you create schemes, it is important that someone familiar with the data review the results. In this particular example, the City scheme chose Dalas as the transformation value for the city of Dallas. Although the values Dalas and Dallas were correctly clustered, you would probably prefer Dallas to be the transformation value.

Note: This example is available in the SAS Sample Library under the name DQSASSCH.

Example 3: Creating Schemes for the QKB

Transformation schemes can be read by SAS and by the DataFlux Data Management Platform software. Generating QKB schemes is advantageous when you want to use DataFlux Data Management Studio to edit the schemes. The following example generates three schemes in QKB scheme file format. Note that the locale ENUSA is assumed to be loaded into memory as part of the locale list.

This example is available in the SAS Sample Library under the name DQQKBSCH.

/* Create filerefs with required suffixes. */
filename city 'c:\my schemes\city.sch.qkb';
filename state 'c:\my schemes\state.sch.qkb';
filename org 'c:\my schemes\org.sch.qkb';

/* Create the input data set. */
data vendors;
  input city $char17. state $char22. company $char36.;
datalines;
  Detroit          MI                     Ford Motor
  Dallas           Texas                  Wal-mart Inc.
  Washington      District of Columbia   Federal Reserve Bank

  Washington      District of Columbia   Federal Reserve Bank

/* See Example 1: Creating an Analysis Data Set for the full data set. */
Example 4: Applying Schemes

In this example, the APPLY statement generates cleansed data in the VENDORS_OUT data set. All schemes are applied before the result is written into the output data set. The match definition for Organization is assumed to be in the QKB used for this code. The locale ENUSA is assumed to be loaded into memory as part of the locale list.

Note: This example is available in the SAS Sample Library under the name DQAPPLY.

/* Create filerefs with required suffixes, and place them in an existing subdirectory. */
filename city 'c:\my schemes\city.sch.qkb';
filename state 'c:\my schemes\state.sch.qkb';
filename org 'c:\my schemes\org.sch.qkb';

/* Create the input data set. */
data vendors;
    input city $char16. state $char22. company $char34.;
datalines;
Detroit     MI        Ford Motor
Dallas      Texas     Wal-mart Inc.
Washington District of Columbia Federal Reserve Bank
SanJose     CA        Wal mart
New York    New York  Ernst & Young
Virginia Bch VA        TRW INC - Space Defense
Dallas      TX        Walmart Corp.
San Francisco California The Jackson Data Corp.
New York    NY        Ernst & Young
Washington District of Columbia Federal Reserve Bank 12th District
New York    N.Y.      Ernst & Young
San Francisco CA        Jackson Data Corporation
Atlanta    GA        Farmers Insurance Group
RTP        NC        Kaiser Permantente
New York    NY        Ernst and Young
Virginia Beach VIRGINIA TRW Space & Defense
Detroit    Michigan  Ford Motor Company
San Jose    CA        Jackson Data Corp
Washington District of Columbia Federal Reserve Bank
Atlanta    GEORGIA   Target
; run;
proc dqscheme data=vendors out=vendors_out qkb;
create matchdef='City'
   var=city
   scheme=city
   locale='ENUSA';
create matchdef='State/Province'
   var=state
   scheme=state
   locale='ENUSA';
create matchdef='Organization'
   var=company
   scheme=org
   locale='ENUSA';
apply var=city scheme=city;
apply var=state scheme=state;
apply var=company scheme=org;
run;

title 'Result after applying all three QKB format schemes';
proc print data=vendors_out;
run;

**Output 9.2**  PROC PRINT Output for DQAPPLY

<table>
<thead>
<tr>
<th>Obs</th>
<th>city</th>
<th>state</th>
<th>company</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Detroit</td>
<td>MI</td>
<td>Ford Motor</td>
</tr>
<tr>
<td>2</td>
<td>Dallas</td>
<td>TX</td>
<td>Wal mart</td>
</tr>
<tr>
<td>3</td>
<td>Washington</td>
<td>District of Columbia</td>
<td>Federal Reserve Bank</td>
</tr>
<tr>
<td>4</td>
<td>San Jose</td>
<td>CA</td>
<td>Wal mart</td>
</tr>
<tr>
<td>5</td>
<td>New York</td>
<td>NY</td>
<td>Ernst &amp; Young</td>
</tr>
<tr>
<td>6</td>
<td>Virginia Bch</td>
<td>VA</td>
<td>TRW INC - Space Defense</td>
</tr>
<tr>
<td>7</td>
<td>Dallas</td>
<td>TX</td>
<td>Wal mart</td>
</tr>
<tr>
<td>8</td>
<td>San Francisco</td>
<td>CA</td>
<td>Jackson Data Corp</td>
</tr>
<tr>
<td>9</td>
<td>New York</td>
<td>NY</td>
<td>Ernst &amp; Young</td>
</tr>
<tr>
<td>10</td>
<td>Washington</td>
<td>District of Columbia</td>
<td>Federal Reserve Bank</td>
</tr>
<tr>
<td>11</td>
<td>New York</td>
<td>NY</td>
<td>Ernst &amp; Young</td>
</tr>
<tr>
<td>12</td>
<td>San Francisco</td>
<td>CA</td>
<td>Jackson Data Corp</td>
</tr>
<tr>
<td>13</td>
<td>Atlanta</td>
<td>GA</td>
<td>Farmers Insurance Group</td>
</tr>
<tr>
<td>14</td>
<td>RTP</td>
<td>NC</td>
<td>Kaiser Permanente</td>
</tr>
<tr>
<td>15</td>
<td>New York</td>
<td>NY</td>
<td>Ernest and Young</td>
</tr>
<tr>
<td>16</td>
<td>Virginia Bch</td>
<td>VA</td>
<td>TRW Space &amp; Defense</td>
</tr>
<tr>
<td>17</td>
<td>Detroit</td>
<td>MI</td>
<td>Ford Motor</td>
</tr>
<tr>
<td>18</td>
<td>San Jose</td>
<td>CA</td>
<td>Jackson Data Corp</td>
</tr>
<tr>
<td>19</td>
<td>Washington</td>
<td>District of Columbia</td>
<td>Federal Reserve Bank</td>
</tr>
<tr>
<td>20</td>
<td>Atlanta</td>
<td>GA</td>
<td>Target</td>
</tr>
</tbody>
</table>
Details

Note that the SCHEME= arguments in this case reference the three filerefs assigned above. All of the schemes will be created, and then they will all be applied, to produce the result in the VENDORS_OUT data set.

Note that you do not specify a locale when you apply a scheme. The application of a scheme simply compares an input value from your input data set against the values in the scheme, so no locale processing is done.

Also note that the schemes are being created and applied within the same step, but only for sample purposes. It is not recommended that you create and then immediately apply a scheme without reviewing the scheme first. If you are using QKB format schemes, the most common approach would be to create, review, and edit your schemes from within SAS Data Integration Studio or DataFlux Data Management Studio. After you review the schemes, you can apply them with the APPLY statement of PROC DQSCHEME (shown here), or with the DQSCHEMAPPLY function or CALL routine.

SCHEME_LOOKUP= EXACT (the default) specifies that the value in the scheme replaces the input value in the output data set. This occurs when an exact match is found between the input value and a DATA value in the scheme. When you use the default scheme apply mode MODE=PHRASE, each input value is compared to the DATA values in the scheme.
%DLOAD AUTOCALL Macro
Sets system option values and loads locales into memory.

**Syntax**

```
%DLOAD option(s);
```

**Summary of Optional Arguments**

status information
```
DQINFO=0 | 1
```

**Required Arguments**

```
DQSETUPLOC= "path-specification"
```

specifies the location (root directory) of the Quality Knowledge Base. The Quality Knowledge Base contains the specified locales.

"path-specification" contains a series of selections that the system follows to reach a specified folder.

*Windows Specifics*

the path specification identifies the root directory of the Quality Knowledge Base.

*UNIX Specifics*

the path specification identifies the root directory of the Quality Knowledge Base.
DQLOCALE=(*locale-1, locale-2, …*)

specifies an ordered list of locales to load into memory.

The %DQLOAD AUTOCALL macro looks for a match only among supported locales. If the requested locale is not supported, it returns an "invalid locale name" error.

Optional Argument

DQINFO=0 | 1

specifies that when DQINFO=1 the additional information that is generated about the status of the locale load operation is written to the SAS log.

Default 0

Details

Specify the %DQLOAD AUTOCALL macro at the beginning of each data cleansing program. This ensures that the proper list and order of locales is loaded into memory before you cleanse data. This loading prevents the use of an unintended default locale or locale list.

Specify the %DQLOAD macro before data cleansing, instead of at SAS invocation, using an AUTOEXEC or configuration file, to preserve memory and shorten the duration of the SAS invocation. Doing so is particularly beneficial when the SAS session is not used to run data cleansing programs.

It is strongly suggested that you use only the %DQLOAD macro to set the value of the DQLOCALE= system option. Setting the value of this system option by the usual means (such as an OPTIONS statement) does not load the specified locales into memory. Not loading locales into memory can lead to the use of an unintended locale. For the same reason, it is not recommended that you set the DQLOCALE= system option at SAS invocation using a configuration file or AUTOEXEC.

In addition to setting the DQLOCALE= system option, the %DQLOAD macro also sets the DQSETUPLOC= system option (if that value is not set by default at your site). When SAS is installed, the value of the DQSETUPLOC= option is set to point to the default location of the sample Quality Knowledge Base.

Example

The following example uses the DQLOCALE option to specify an ordered list of locales to load into memory. DQSETUPLOC specifies the location of the Quality Knowledge Base.

```
%DQLOAD(DQLOCALE=(ENUSA DEDEU), DQSETUPLOC='/sas/dqc/QKBLoc');
```

%DQPUTLOC AUTOCALL Macro

Displays current information about a specified locale in the SAS log.

Tip: Specifying no parameters displays the full report for the default locale.

Syntax

```
%DQPUTLOC option(s);
```
Summary of Optional Arguments

lists related parse definition
   PARSEDEFN=0 | 1

shortens length of log
   SHORT=0 | 1

specifies the local of interest
   locale

Optional Arguments

locale
   specifies the locale of interest. The value can be a locale name in quotation marks or an expression that evaluates to a locale name. The specified locale must have been loaded into memory as part of the locale list.

   Default        Locale is the first locale in the locale list.
   Requirement    Locale must be loaded into memory before this macro is called.
   Tip            Specifying no parameters displays the full report for the default locale.
   See            “DQSETUPLOC= System Option” on page 132 for additional information.

PARSEDEFN=0 | 1
   lists with each gender analysis definition and each match definition and the related parse definition, if the parse definition exists.
   Specify PARSEDEFN = 0 if you do not want to display the related parse definitions.
   Specify PARSEDEFN = 1 to display the related parse definition.

   Default  1

SHORT=0 | 1
   Shortens the length of the entry in the SAS log.
   Specify SHORT =0 to display the descriptions of how the definitions are used.
   Specify SHORT =1 to remove the descriptions of how the definitions are used.

   Default  0

Details

The %DQPUTLOC AUTOCALL macro displays the contents of the specified locale in the SAS log. Locale contents include all definitions, parse tokens, related functions, and the names of the parse definitions that are related to each match definition. Knowing the related parse definitions enables the creation of parsed character values. See “DQPARESTOKENPUT Function” on page 114 for additional information.

It also enables the creation of match codes for parsed character values. See “DQMATCHPARSED Function” on page 106 for additional information.
Load the specified locale into memory with `%DQLOAD` before you submit `%DQPUTLOC`.

**Example**

This example displays in the SAS log definitions, related parse definitions, and related SAS Data Quality Server functions for the ENUSA locale.

```
%dqputloc(enusa);
```

**See Also**

**References**

- “DQLOCALEINFOGET Function” on page 102
- “DQLOCALEINFOLIST Function” on page 102

---

**%DQUNLOAD AUTOCALL Macro**

Unloads all locales to increase the amount of free memory.

**Requirement:** After unloading locales from memory, load locales with the %DQLOAD AUTOCALL macro before running any data cleansing programs.

**Syntax**

```sas
%DQUNLOAD;
```

**Details**

The `%DQUNLOAD AUTOCALL` macro unloads all locales that are currently loaded into memory. After unloading memory, be sure to load locales again with the `%DQLOAD AUTOCALL` macro before running any data cleansing programs.

**Macro Resources**

**See:**

- SAS Macro Language: Reference
- SAS Language Reference: Concepts
Chapter 11
Functions and CALL Routines

Overview .................................................. 74
Functions Listed Alphabetically ......................... 74
Functions Listed by Category ............................. 76
DataFlux Data Management Server Functions .......... 76
Case Functions ........................................... 76
Gender Analysis, Locale Guessing, and Identification Functions ............................ 77
Matching Functions ....................................... 77
Parsing Functions ......................................... 77
Extraction Functions ....................................... 78
Pattern Analysis Functions ................................. 78
Reporting Functions ....................................... 78
Scheme Functions and CALL Routines ................. 79
Standardization Functions ................................. 79
Dictionary .................................................. 80
DMSRVBATCHJOB Function .............................. 80
DMSRVCOPYLOG Function ................................ 81
DMSRDELETELOG Function ............................... 83
DMSRVJOBSTATUS Function .............................. 84
DMSRVKILLJOB Function .................................. 86
DMSRVPPROFILEJOB Function ......................... 87
DMSRVUSER Function ..................................... 89
DMSRVER Function ......................................... 90
DQCASE Function ......................................... 91
DQEXTINFOGET Function .................................. 92
DQEXTRACT Function ...................................... 93
DQEXTOKENGET Function .................................. 94
DQEXTOKENPUT Function .................................. 95
DQGENDER Function ...................................... 96
DQGENDERINFOGET Function ............................. 97
DQGENDERPARSED Function .............................. 98
DQIDENTIFY Function ..................................... 99
DQLCALEGUESS Function ................................ 100
DQLCALEINFOGET Function .............................. 102
DQLCALEINFOLIST Function ............................. 102
DQMATCH Function ....................................... 104
DQMATCHINFOGET Function .............................. 105
DQMATCHPARSED Function ............................... 106
DQOPTSURFACE Function ................................ 107
DQPARSE Function ....................................... 108
DQPARSE CALL Routine ................................... 109
Overview

The functions and CALL routines in the SAS Data Quality Server software enable you to cleanse data and access DataFlux Data Management Servers.

The functions and CALL routines are listed alphabetically and by category. Each function and CALL routine has a link to a detailed description and syntax.

Note: The SAS Data Quality Server functions and CALL routines are available in the Expression Builder of SAS Data Integration Studio software and SAS Enterprise Guide software.

Functions Listed Alphabetically

- The “DMSRVBATCHJOB Function” on page 80 runs a DataFlux data or process job on a DataFlux Data Management Server and returns a job identifier.
- The “DMSRVCOPYLOG Function” on page 81 copies a job's log file from a DataFlux Data Management Server.
- The “DMSRDELETELOG Function” on page 83 deletes a job's log file from a DataFlux Data Management Server.
- The “DMSRVJOBSTATUS Function” on page 85 returns the status of a job that was submitted to a DataFlux Data Management Server.
- The “DMSRVKILLJOB Function” on page 86 terminates a job that is running on a DataFlux Data Management Server.
- The “DMSRVPROFILEJOB Function” on page 87 generates a profile from a Data Management Server repository.
- The “DMSRVUSER Function” on page 89 registers credentials (user name and password) on a DataFlux Data Management Server and returns a value to indicate the success or failure of credential storage.
- The “DMSRVER Function” on page 90 returns the version of the DataFlux Data Management Server.
- The “DQCASE Function” on page 91 returns a character value with standardized capitalization.
• The “DQCASE Function” on page 91 returns the token names in an extraction definition.
• The “DQCASE Function” on page 91 returns an extracted character value.
• The “DQCASE Function” on page 91 returns a token from an extraction character value.
• The “DQCASE Function” on page 91 inserts a token into an extraction character value and returns the updated extraction character value.
• The “DQGENDER Function” on page 96 returns a gender determination from the name of an individual.
• The “DQGENDERINFOGET Function” on page 97 returns the name of the parse definition that is associated with a specified gender analysis definition.
• The “DQGENDERPARSED Function” on page 98 returns a gender determination from the parsed name of an individual.
• The “DQIDENTIFY Function” on page 99 returns a category name from a character value.
• The “DQLOCALEGUESS Function” on page 100 returns the name of the locale that is most likely represented by a character value.
• The “DQLOCALEINFOGET Function” on page 102 returns information about locales.
• The “DQMATCHPARSED Function” on page 106 returns the names of the definitions in a locale and returns a count of those definitions.
• The “DQMATCH Function” on page 104 returns a match code from a character value.
• The “DQMATCHINFOGET Function” on page 105 returns the name of the parse definition that is associated with a match definition.
• The “DQMATCHPARSED Function” on page 106 returns a match code from a parsed character value.
• The “DQOPTSURFACE Function” on page 107 reveals or hides non-surfaced definitions.
• The “DQPARSE CALL Routine” on page 109 returns a parsed character value and a status flag.
• The “DQPARSE Function” on page 108 returns a parsed character value.
• The “DQPARSEINFOGET Function” on page 110 returns the token names for the specified parse definition.
• The “DQPARSEINPUTLEN Function” on page 111 sets the default length of parsed input, and returns a string indicating its previous value.
• The “DQPARSERESLIMIT Function” on page 112 sets a limit on resources consumed during parsing.
• The “DQPARSESCORDEPTH Function” on page 113 specifies how deeply to search for the best parsing score.
• The “DQPARGETOKENGET Function” on page 113 returns a token from a parsed character value.
• The “DQPARGETOKENPUT Function” on page 114 inserts a token into a parsed character value and returns the updated parsed character value.
Functions Listed by Category

DataFlux Data Management Server Functions

- The “DMSRVBATCHJOB Function” on page 80 runs a DataFlux Data Management Studio process or data job on a DataFlux Data Management Server and returns a job identifier.
- The “DMSRVCOPYLOG Function” on page 81 copies a job's log file from a DataFlux Data Management Server.
- The “DMSRDELETELOG Function” on page 83 deletes a job's log file from a DataFlux Data Management Server.
- The “DMSRVJOBSTATUS Function” on page 84 returns the status of a job that was submitted to a DataFlux Data Management Server.
- The “DMSRVKILLJOB Function” on page 86 terminates a job that is running on a DataFlux Data Management Server.
- The “DMSRVPROFILEJOB Function” on page 87 generates a profile from a repository on a DataFlux Data Management Server.
- The “DMSRUSER Function” on page 89 registers credentials (user name and password) on a DataFlux Data Management Server and returns a value to indicate the success or failure of credential storage.
- The “DMSRVER Function” on page 90 returns the version of the DataFlux Data Management Server.

Case Functions

- The “DQCASE Function” on page 91 returns a character value with standardized capitalization.
The “DQSTANDARDIZE Function” on page 125 returns a character value after standardizing casing, spacing, and format, and applying a common representation to certain words and abbreviations.

**Gender Analysis, Locale Guessing, and Identification Functions**

The gender analysis, locale guessing, and identification functions return information that is determined from the content of an input character value.

- The “DQGENDER Function” on page 96 returns a gender determination from the name of an individual.
- The “DQGENDERINFOGET Function” on page 97 returns the name of the parse definition that is associated with a specified gender analysis definition.
- The “DQGENDERPARSED Function” on page 98 returns a gender determination from the parsed name of an individual.
- The “DQIDENTIFY Function” on page 99 returns a category name from a character value.
- The “DQLocateGUESS Function” on page 100 returns the name of the locale that is most likely represented by a character value.
- The “DQLOCALEINFOGET Function” on page 102 returns information about locales.
- The “DQMATCHPARSED Function” on page 106 returns the names of the definitions in a locale and returns a count of those definitions.

**Matching Functions**

- The “DQMATCH Function” on page 104 returns a match code from a character value.
- The “DQMATCHINFOGET Function” on page 105 returns the name of the parse definition that is associated with a match definition.
- The “DQMATCHPARSED Function” on page 106 returns a match code from a parsed character variable.

**Parsing Functions**

- The “DQGENDERINFOGET Function” on page 97 returns the name of the parse definition that is associated with a specified gender analysis definition.
- The “DQGENDERPARSED Function” on page 98 returns a gender determination from the parsed name of an individual.
- The “DQMATCHPARSED Function” on page 106 returns a match code from a parsed character value.
- The “DQPARSE CALL Routine” on page 109 returns a parsed character value and a status flag.
- The “DQPARSE Function” on page 108 returns a parsed character value.
- The “DQPARSEINFOGET Function” on page 110 returns the token names for the specified parse definition.
• The “DQPARSEINPUTLEN Function” on page 111 sets the default length of parsed input. DQPARSEINPUTLEN also returns a string indicating its previous value.

• The “DQPARSERESLIMIT Function” on page 112 sets a limit on resources consumed during parsing.

• The “DQPARSESCORDEPTH Function” on page 113 specifies how deeply to search for the best parsing score.

• The “DQPARSETOKENGET Function” on page 113 returns a token from a parsed character value.

• The “DQPARSETOKENPUT Function” on page 114 inserts a token into a parsed character value and returns the updated parsed character value.

**Extraction Functions**

• The “DQEXTINFOGET Function” on page 92 returns the token names in an extraction definition.

• The “DQEXTRACT Function” on page 93 returns an extracted character value.

• The “DQEXTTOKENGET Function” on page 94 returns a token from an extraction character value.

• The “DQEXTTOKENPUT Function” on page 95 inserts a token into an extraction character value and returns the updated extraction character value.

**Pattern Analysis Functions**

The “DQPATTERN Function” on page 116 returns a pattern analysis from an input character value.

**Reporting Functions**

• The “DQGENDER Function” on page 96 returns a gender determination from the name of an individual.

• The “DQGENDERPARSED Function” on page 98 returns a gender determination from the parsed name of an individual.

• The “DQGENDERINFOGET Function” on page 97 returns the name of the parse definition that is associated with a specified gender analysis definition.

• The “DQIDENTIFY Function” on page 99 returns a category name from a character value.

• The “DQLOCALEGUESS Function” on page 100 returns the name of the locale that is most likely represented by a character value.

• The “DQLOCALEINFOGET Function” on page 102 returns information about locales.

• The “DQMATCHPARSED Function” on page 106 returns the names of the definitions in a locale and returns a count of those definitions.

• The “DQMATCH Function” on page 104 returns a match code from a character value.

• The “DQMATCHINFOGET Function” on page 105 returns the name of the parse definition that is associated with a match definition.
The “DQMATCHPARSED Function” on page 106 returns a match code from a parsed character value.

The “DQPARSE CALL Routine” on page 109 returns a parsed character value and a status flag.

The “DQPARSE Function” on page 108 returns a parsed character value.

The “DQPARSEINFOGET Function” on page 110 returns the token names for the specified parse definition.

The “DQPARESETOKENGET Function” on page 113 returns a token from a parsed character value.

The “DQPARESETOKENPUT Function” on page 114 inserts a token into a parsed character value and returns the updated parsed character value.

The “DQPATTERN Function” on page 116 returns a pattern analysis from an input character value.

The “DQSCHEMEAPPLY CALL Routine” on page 120 applies a scheme and returns a transformed value and a transformation flag.

The “DQSCHEMEAPPLY Function” on page 117 applies a scheme and returns a transformed value after applying a scheme.

The “DQSTANDARDIZE Function” on page 125 returns a character value after standardizing casing, spacing, and format, and applying a common representation to certain words and abbreviations.

The “DQTOKEN Function” on page 126 returns a token from a character value.

The “DQVER Function” on page 127 returns the version of the SAS Data Quality engine.

The “DQVERQKB Function” on page 127 returns the version of the currently loaded Quality Knowledge Base.

**Scheme Functions and CALL Routines**

- The “DQSCHEMEAPPLY Function” on page 117 applies a scheme and returns a transformed value.
- The “DQSCHEMEAPPLY CALL Routine” on page 120 applies a scheme and returns a transformed value and a transformation flag.

**Standardization Functions**

- The “DQCASE Function” on page 91 returns a character value with standardized capitalization.
- The “DQSTANDARDIZE Function” on page 125 returns a character value after standardizing the casing, spacing, and format, and after applying a common representation to certain words and abbreviations.
Dictionary

DMSRVBATCHJOB Function

Runs a DataFlux data or process job on a DataFlux Data Management Server and returns a job identifier.

Valid in: DATA step, PROC SQL, and SAS Component Language

Requirements: If specified, the locale must be loaded into memory as part of the locale list.

The character variable that receives the return value must have a minimum length of 52.

Syntax

DMSRVBATCHJOB(job-name, host, port <,parameter-list>)

Required Arguments

**job-name**

the DataFlux Data Management Studio job or process as it exists on the specified DataFlux Data Management Studio Server.

**host**

identifies the host of the DataFlux Data Management Server. The host name can be a variable or a literal string. The literal string or the value of the variable is the URL of the server in single quotation marks.

When DataFlux Data Management Server is secured with SSL, the URL must use https instead of http (for example, https://myhost.unx.com).

Interaction

If a zero-length string is entered for the host argument, then the value localhost is used.

**Example**

```
/* Incorrect use of function arguments */
dmsrvBatchJob('jobname');

/* Localhost is used for the host */
dmsrvBatchJob(jobname, '', 21036);

/* Correct */
dmsrvBatchJob(jobname, 'http://myhost.unx.com', 21036);
```

**port**

identifies the port through which the host communicates with the DataFlux Data Management Server.

Interaction

If the value specified is less than or equal to 0, then port number 21036 is used with SOAP, or port number 21037 is used when using Wireline.

**Example**

```
/* Incorrect use of function arguments */
dmsrvBatchJob('jobname');
```
/* Port 21036 or 21037 is used */
dmsrvBatchJob('jobname', 'http://myhost.unx.com', 0);

/* Correct */
dmsrvBatchJob('jobname', 'http://myhost.unx.com', 21036);

Optional Argument

`parameter-list`  
the variable list of name and value pairs, where each name and value pair in the list must be defined as an input to the job.

Details

The DMSRVBATCHJOB function returns a job-identifier. The return value is either a job identifier of up to 52 characters or the value MISSING. Use the job identifier in subsequent function calls to manage the job, using DMSRVJOBSTATUS, DMSRVCOPYLOG, DMSRVDELETELOG, and DMSRVKILLJOB.

- You can specify any number of macro value pairs.

Example: DMSRVBATCHJOB Function

The following example runs a job on a DataFlux Data Management Server.

```sas
data _null_
  jobid = dmsrvBatchJob  ('myjob.djf', 'http://myhost.unx.com', 21036);
run;
```

See Also

Functions

- “DMSRVCOPYLOG Function” on page 81
- “DMSRVDELETELOG Function” on page 83
- “DMSRVJOBSTATUS Function” on page 84

DMSRVCOPYLOG Function

Copies a job's log file from a DataFlux Data Management Server to a local host.

Valid in: DATA step, PROC SQL, and SAS Component Language

Syntax

```
DMSRVCOPYLOG(job-ID, host, port, filename)
```

Required Arguments

`job-ID`  
identifies the job that is submitted to a DataFlux Data Management Server. The identifier is previously returned by a function such as DMSRVBATCHJOB.
host
identifies the host of the DataFlux Data Management Server. The host name can be a variable or a literal string. The literal string or the value of the variable is the URL of the server in single quotation marks.

When DataFlux Data Management Server is secured with SSL, the URL must use https instead of http (for example, https://myhost.unx.com).

Interaction
If a zero-length string is entered for the host argument, then the value localhost is used.

Example
/* Incorrect use of function arguments */
dmsrvCopyLog('jobid');

/* Localhost is used for the host */
dmsrvCopyLog('jobid', '', 21036, 'filename');

/* Correct */
dmsrvCopyLog('jobid', 'http://myhost.unx.com', 21036, 'filename');

port
identifies the port through which the host communicates with the DataFlux Data Management Server.

Interaction
If the value specified is less than or equal to 0, then port number 21036 is used with SOAP, or port number 21037 is used when using Wireline.

Example
/* Incorrect use of function arguments */
dmsrvCopyLog('jobid');

/* Port 21036 or 21037 is used */
dmsrvCopyLog('jobid', 'http://myhost.unx.com', 0, 'filename');

/* Correct */
dmsrvCopyLog('jobid', 'http://myhost.unx.com', 21036, 'filename');

filename
identifies where the log file is copied on the local host.

Details
To capture log information for a particular job, use the DMSRVJOBSTATUS function to ensure that the job is finished before you copy the log.

Return values are 0 (log copied successfully) or 1 (log failed to copy).

Example: DMSRVCOPYLOG Function
The following example copies a log file from a DataFlux Data Management Server. The log file is generated when the server runs a job. The job identifier is returned in the function that runs the job.

copyrc= dmsrvCopyLog(jobid,'http://myhost.unx.com', 5001,'dmServer1.log');
DMSRVDELETELOG Function

Deletes a job’s log file from a DataFlux Data Management Server.

**Valid in:** DATA step, PROC SQL, and SAS Component Language

### Syntax

```sas
DMSRVDELETELOG(job-ID, host, port)
```

### Required Arguments

**job-ID**

identifies the job submitted to a DataFlux Data Management Server. The identifier is set by a function such as DMSRVBATCHJOB.

**host**

identifies the host of the DataFlux Data Management Server. The host name can be a variable or a literal string. The literal string or the value of the variable is the URL of the server in single quotation marks.

When DataFlux Data Management Server is secured with SSL, the URL must use **https** instead of **http** (for example, https://myhost.unx.com).

**Interaction**

If a zero-length string is entered for the `host` argument, then the value `localhost` is used.

**Example**

```sas
/* Incorrect use of function arguments */
DMSRVDELETELOG('jobid');

/* Localhost is used for the host */
DMSRVDELETELOG('jobid', '', 21036);

/* Correct */
DMSRVDELETELOG('jobid', 'http://myhost.unx.com', 21036);
```

**port**

identifies the port through which the host communicates with the DataFlux Data Management Server.

**Interaction**

If the value specified is less than or equal to 0, then port number 21036 is used with SOAP, or port number 21037 is used when using Wireline.

**Example**

```sas
/* Incorrect use of function arguments */
DMSRVDELETELOG('jobid');
```
Details

The log file is created after the job terminates. Use DMSRVJOBSTATUS to ensure that
the log file is available for deletion.

- DMSRVDELETELOG does not delete local copies of the job's log file.
- Return values are 0 (log deleted successfully) or 1 (log failed to delete).

Example: DMSRVDELETELOG FUNCTION

The following example deletes a log file from a DataFlux Data Management Server. The
log file is created when the server runs a job. The job identifier is returned in the
function that runs the job.

```sql
delrc= dmsrvDeleteLog(jobid,'http://myhost.unx.com', 5001);
```

See Also

Functions

- “DMSRVBATCHJOB Function” on page 80
- “DMSRVCOPYLOG Function” on page 81
- “DMSRVJOBSTATUS Function” on page 84

DMSRVJOBSTATUS Function

Returns the status of a job that was submitted to a DataFlux Data Management Server.

Valid in: DATA step, PROC SQL, and SAS Component Language

Syntax

DMSRVJOBSTATUS(job-ID, host, port, time-out, interval)

Required Arguments

**job-ID**

identifies the job that was submitted to a DataFlux Data Management Server. The
identifier is previously set by a function such as DMSRVBATCHJOB.

**host**

identifies the host of the DataFlux Data Management Server. The host name can be a
variable or a literal string. The literal string or the value of the variable is the URL of
the server in single quotation marks.

When DataFlux Data Management Server is secured with SSL, the URL must use
https instead of http (for example, https://myhost.unx.com).
Interaction

If a zero-length string is entered for the host argument, then the value localhost is used.

Example

/* Incorrect use of function arguments */
    dmsrvJobStatus('jobid');

/* Localhost is used for the host */
    dmsrvJobStatus('jobid', '', 21036, 20, 5);

/* Correct */
    dmsrvJobStatus('jobid', 'http://myhost.unx.com', 21036, 20, 5);

port

identifies the port through which the host communicates with the DataFlux Data Management Server.

Interaction

If the value specified is less than or equal to 0, then port number 21036 is used with SOAP, or port number 21037 is used when using Wireline.

Example

/* Incorrect use of function arguments */
    dmsrvJobStatus('jobid');

/* Port 21036 or 21037 is used */
    dmsrvJobStatus('jobid', 'http://myhost.unx.com', 0, 20, 5);

/* Correct */
    dmsrvJobStatus('jobid', 'http://myhost.unx.com', 21036, 20, 5);

time-out

a time in seconds that determines when status information is returned from the host. Valid values are defined as follows:

-1 returns status information about when the job is finished. Return values are 0 (job completed successfully) or 1 (job failed). This value invalidates the interval argument.

0 returns status information immediately. Return values are 0 (job completed successfully), 1 (job failed), or 2 (job running). This value invalidates the interval argument.

greater-than-zero specifies a time limit for the interval argument. If the job is still running after the time-out value, another value is returned only when the job is finished.

interval

the repeat period for the return of status information, within the limit that is imposed by the time-out argument.

Details

Use the DMSRVJOBSTATUS function to return job status information instantly, periodically, or at the completion of the job. With an interval of 20 and a time-out of 60, DMSRVJOBSTATUS returns status information up to four times. After 60 seconds, the last return value is provided at the completion of the job.
Return values are 0 (job completed successfully), 1 (job failed), or 2 (job running).

Example: DMSRVJOBSTATUS Function

The following example returns a status number for a job that ran or is running on a DataFlux Data Management Server. The job identifier was returned by the function that ran the job. Status information is returned in 20 seconds or less, depending on the termination of the job. Job status is checked every 5 seconds.

```plaintext
status= dmsrvJobStatus(jobid,'http://myhost.unx.com', 5001, 20, 5);
```

See Also

Functions

- “DMSRVBATCHJOB Function” on page 80
- “DMSRVDELETELOG Function” on page 83
- “DMSRVKILLJOB Function” on page 86

DMSRVKILLJOB Function

Terminates a job that is running on a DataFlux Data Management Server.

**Valid in:** DATA step, PROC SQL, and SAS Component Language

**Syntax**

```plaintext
DMSRVKILLJOB(job-ID, host, port)
```

**Required Arguments**

- **job-ID**
  
  identifies the job submitted to a DataFlux Data Management Server. The identifier is set by a function such as DMSRVBATCHJOB.

- **host**
  
  identifies the host of the DataFlux Data Management Server. The host name can be a variable or a literal string. The literal string or the value of the variable is the URL of the server in single quotation marks.

  When DataFlux Data Management Server is secured with SSL, the URL must use `https` instead of `http` (for example, `https://myhost.unx.com`).

  Interaction

  If a zero-length string is entered for the `host` argument, then the value `localhost` is used.

**Example**

```plaintext
/* Incorrect use of function arguments */
dmsrvKillJob('jobid');

/* Localhost is used for the host */
dmsrvKillJob('jobid', '', 21036);

/* Correct */
```
port
identifies the port through which the host communicates with the DataFlux Data
Management Server.

Interaction
If the value specified is less than or equal to 0, then port number 21036 is used
with SOAP, or port number 21037 is used when using Wireline.

Example

```c
/* Incorrect use of function arguments */
dmsrvKillJob('jobid');

/* Port 21036 or 21037 is used */
dmsrvKillJob('jobid', 'http://myhost.unx.com', 0);

/* Correct */
dmsrvKillJob('jobid', 'http://myhost.unx.com', 21036);
```

Details
The DMSRVKILLJOB function terminates a job. Use the DMSRVJOBSTATUS
function to determine whether a job is still running. Return values are 0 (job terminated)
or 1 (job failed to terminate).

Example: DMSRVKILLJOB Function

The following example terminates a job that is running on a DataFlux Data Management
Server. The job identifier is returned by the function that ran the job. Status information
is returned in 20 seconds or less, depending on the termination of the job. Job status is
checked every 5 seconds.

```c
killrc= dmsrvKillJob('jobid','http://myhost.unx.com',5001);
```

See Also

Functions

- “DMSRVBATCHJOB Function” on page 80
- “DMSRVJOBSTATUS Function” on page 84

DMSRVPROFILEJOB Function

Generates a profile from a Data Management server repository.

Valid in: DATA step, PROC SQL, and SAS Component Language

Requirement: The character variable that receives the return value must have a minimum length of 52 characters.

Syntax

```c
DMSRVPROFILEJOB(job-name, host, port, append-flag <,description-character>)
```
**Required Arguments**

**job-name**
identifies the DataFlux Data Management Profile job as it exists on the specified DataFlux Data Management Server.

**host**
identifies the host of the DataFlux Data Management Server. The host name can be a variable or a literal string. The literal string or the value of the variable is the URL of the server in single quotation marks.

When DataFlux Data Management Server is secured with SSL, the URL must use **https** instead of **http** (for example, https://myhost.unx.com).

**Interaction**
If a zero-length string is entered for the **host** argument, then the value **localhost** is used.

**Example**

```c
/* Incorrect use of function arguments */
dmsrvProfileJob('jobname');

/* Localhost is used for the host */
dmsrvProfileJob('jobname', '', 21036, 0);

/* Correct */
dmsrvProfileJob('jobname', 'http://myhost.unx.com', 21036, 0);
```

**port**
identifies the port through which the host communicates with the DataFlux Data Management Server.

**Interaction**
If the value specified is less than or equal to 0, then port number 21036 is used with SOAP, or port number 21037 is used when using Wireline.

**Example**

```c
/* Incorrect use of function arguments */
dmsrvProfileJob('jobname');

/* Port 21036 or 21037 is used */
dmsrvProfileJob('jobname', 'http://myhost.unx.com', 0, 1);

/* Correct */
dmsrvProfileJob('jobname', 'http://myhost.unx.com', 21036, 1);
```

**append-flag**
appends or overwrites job results.

0  appends job results below any existing content in the results file.

1  overwrites any existing content in the results file.
Optional Argument

description-character

identifies a character variable whose value describes the current run of the job. The
descriptive text is added either to the top of the results file or above the results that
are appended to the bottom of the results file.

Details

The DMSRVPROFILEJOB function generates a profile from a Data Management server
repository.

Example: DMSRVPROFILEJOB Function

The following example generates a profile from the specified repository.

data _null_;
   jobid = dmsrvProfileJob('myfolder/prof_job',
                    'http://myhost.unx.com', 21036, 1);
   run;

See Also

Functions

• “DMSRVMJOBSTATUS Function” on page 85
• “DMSRVKILLJOB Function” on page 87

DMSRVUSER Function

Registers a user on a DataFlux Data Management Server.

Valid in: DATA step, PROC SQL, and SAS Component Language

Syntax

DMSRVUSER(user-ID, password)

Required Arguments

user-ID

identifies a user-ID according to the registry in a DataFlux Data Management Server.

password

identifies the associated user-ID user according to the registry in the DataFlux Data
Management Server. The password can be plain text or encoded in SAS.

Details

The DMSRVUSER function registers a user on a secure DataFlux Data Management
Server. A return value of zero indicates storage of credentials was successful. A return
value of 1 indicates a failure to store the credentials.
• Call this function as needed in a single DATA step to access different Data Management Servers or to change the registered user credentials within a single Data Management Server.

• If security has not been configured on a DataFlux Data Management Server, then the DMSRVUSER function has no effect.

• Return values are 0 (successful registration of credentials) or 1 (failed to register credentials).

**Example: DMSRVUSER Function**

The following example supplies a user identifier and a password to a secure DataFlux Data Management Server:

```sas
rc = dmsrvUser('dfUser3', 'pwdUser3');
```

---

**DMSRVVER Function**

Returns the version of the DataFlux Data Management Server.

**Valid in:** DATA step, PROC SQL, and SAS Component Language

**Syntax**

```
DMSRVVER(host, port)
```

**Required Arguments**

**host**

identifies the host of the DataFlux Data Management Server. The host name can be a variable or a literal string. The literal string or the value of the variable is the URL of the server in single quotation marks.

When DataFlux Data Management Server is secured with SSL, the URL must use `https` instead of `http` (for example, `https://myhost.unx.com`).

**Interaction**

If a zero-length string is entered for the `host` argument, then the value `localhost` is used.

**Example**

```sas
   /* Localhost is used for the host */
   dmsrvVer('', 21036);
```

**port**

identifies the port through which the host communicates with the DataFlux Data Management Server.

**Interaction**

If the value specified is less than or equal to 0, then port number 21036 is used with SOAP, or port number 21037 is used when using Wireline.

**Example**

```sas
   /* Port 21036 or 21037 is used */
   dmsrvVer('http://myhost.unx.com', 0);
```
Details
The DMSRVVER function takes two arguments, a host name and a port number. If host is not specified, the local host is used. If port is not specified, or if the value is zero or a negative number, the default port number 21036 is used.

DMSRVVER returns a string listing the version number of the integration server, designated by the host and port values.

Example: DMSRVVER Function
The following example sets the value of the version to the character string of the DataFlux Data Management Server, running on machine 'myhost' and communicating with port 19525.

version=dmsrvVer('http://myhost.unx.com', 19525);

See Also
“DMSRVBATCHJOB Function” on page 80

DQCASE Function
Returns a character value with standardized capitalization.

Valid in:   DATA step, PROC SQL, and SAS Component Language
Requirement: The specified locale must be loaded into memory as part of the locale list.

Syntax
DQCASE(char, 'case-definition' <, 'locale'>)

Required Arguments
char
specifies a character constant, variable, or expression that contains the value that is transformed, according to the specified case definition.

case-definition
the character constant, variable, or expression to search. The definition must be in the locale that is used. If the value of char is represented by a case definition, the use of that definition is recommended over the generic case definition.

If the value of char is a street address and you are using the ENUSA locale, the recommended case definition is PROPER–ADDRESS. This is used instead of the generic case definition PROPER.

Optional Argument
locale
specifies a character constant, variable, or expression that contains the locale name.

Default The default locale is the first locale in the locale list. If no value is specified, the default locale is used.
Details
The DQCASE function operates on any character content, such as names, organizations, and addresses.

Example: DQCASE Function
The following example standardizes the capitalization and spacing with the PROPER case definition in the ENUSA locale.

\[
\text{orgname}=\text{dqCase}("\text{BILL'S PLUMBING & HEATING}", \text{Proper}, \text{ENUSA})
\]

After this function call, the value of ORGNAME is Bill's Plumbing & Heating.

---

DQEXTINFOGET Function
Returns the token names in an extraction definition.

Valid in: DATA step, PROC SQL, and SAS Component Language

Required Argument

\[\text{DQEXTINFOGET('extraction-definition' ,locale')}\]

Required Argument

\[\text{extraction-definition}\]

specifies the name of the extraction definition. The definition must exist in the locale that is used.

Optional Argument

\[\text{locale}\]

specifies a character constant, variable, or expression that contains the locale name.

Default

The default locale is the first locale in the locale list. If no value is specified, the default locale is used.

Details
The DQEXTINFOGET function returns the names of the tokens that can be inserted into character values using the DQEXTTOKENPUT function.

Example: DQEXTINFOGET Function
The following example returns the token names for the extraction definition email in the locale ENUSA and displays the token names in the SAS log.

\[
\text{tokenNames}=\text{dqExtInfoGet('e-mail', 'ENUSA')};
\text{put tokenNames;}
\]

After this function call, the value of TOKENNAMES is Mailbox, Sub-Domain, Top-Level Domain, which are the names of the three tokens in this extraction definition.
See Also

Functions

• “DQEXTTOKENGET Function” on page 94
• “DQEXTTOKENPUT Function” on page 95
• “DQEXTRACT Function” on page 93

DQEXTRACT Function

Returns an extracted character value.

Valid in: DATA step, PROC SQL, and SAS Component Language

Restriction: Always use the DQEXTTOKENGET function to retrieve tokens from extracted values. To extract tokens from values that do not contain delimiters, use the DQTOKEN function.

Requirement: If specified, the locale must be loaded into memory as part of the locale list.

Syntax

DQEXTRACT('extraction-string', 'extraction-definition' [,locale])

Required Arguments

extraction-string

the value that is extracted according to the specified extraction definition. The value must be the name of a character variable, or a character value in quotation marks. Also valid, an expression that evaluates to a variable name or quoted value.

extraction-definition

the name of the extraction definition. The definition must exist in the locale that is used.

Optional Argument

locale

specifies a character constant, variable, or expression that contains the locale name.

Default The first locale in the locale list. If no value is specified, the default locale is used.

Details

The DQEXTRACT function returns an extracted character value. The return value contains delimiters that identify the elements in the value that correspond to the tokens that are enabled by the extraction definition. The delimiters in the value allow functions such as DQEXTTOKENGET to access the elements in the value based on specified token names.
Example

The following example extracts the name of an individual. Then the DQEXTTOKENGET function returns the values of two of the tokens.

```plaintext
extValue=dqExtract('Mr. James Joseph Westly', 'NAME', 'ENUSA');
prefix=dqExtTokenGet(extValue, 'Name Prefix', 'NAME', 'ENUSA');
given=dqExtTokenGet(extValue, 'Given Name', 'NAME', 'ENUSA');
```

After these function calls, the value of PREFIX is Mr. and the value of GIVEN is James.

See Also

- “DQEXTINFOGET Function” on page 92
- “DQTOKEN Function” on page 126

DQEXTTOKENGET Function

Returns a token from an extraction character value.

**Valid in:** DATA step, PROC SQL, and SAS Component Language

**Restriction:** Do not attempt to retrieve tokens from extraction values using any means other than the DQEXTTOKENGET function.

**Requirement:** If specified, the locale must be loaded into memory as part of the locale list.

**Syntax**

```plaintext
DQEXTTOKENGET(extraction-char, 'token', 'extraction-definition' <, 'locale'>)
```

**Required Arguments**

**extraction-char**

specifies a character constant, variable, or expression that contains the value that is the extraction character value from which the value of the specified token is returned.

To determine how the extraction definition inserts delimiters, use the DQEXTINFOGET function.

**token**

the name of the token that is returned from the extraction value. The token must be enabled by the specified extraction definition

**extraction-definition**

the name of the extraction definition. The definition must exist in the locale that is used. The extraction definition must be the same as the extraction definition that originally extracted the EXTRACTION-CHAR value.

**Optional Argument**

**locale**

specifies a character constant, variable, or expression that contains the locale name.

**Default**

The default locale is the first locale in the locale list. If no value is specified, the default locale is used.
Details
The DQEXTTOKENGET function returns the value of the specified token from a previously extracted character value.

Example
The following example extracts a character value with the DQEXTRACT function and extracts two of the tokens with the DQEXTTOKENGET function.

```sql
extValue=dqExtract('Mr. James Joseph Westly', 'NAME', 'ENUSA');
prefix=dqExtTokenGet(extValue, 'Name Prefix', 'NAME', 'ENUSA');
given=dqExtTokenGet(extValue, 'Given Name', 'NAME', 'ENUSA');
```

After these function calls, the value of `prefix` is `Mr.` and the value of `given` is `James`.

See Also
Functions
- “DQEXTRACT Function” on page 93
- “DQEXTINFOGET Function” on page 92
- “DQTOKEN Function” on page 126

---

**DQEXTTOKENPUT Function**
Inserts a token into an extraction character value and returns the updated extraction character value.

**Valid in:** DATA step and SAS Component Language

**Requirement:** If specified, the locale must be loaded into memory as part of the locale list.

**Syntax**

```
DQEXTTOKENPUT('extraction-char', 'token-value', 'token-name', 'extraction-definition' <, 'locale'>)
```

**Required Arguments**

- `extraction-char`:
  specifies a character constant, variable, or expression that contains the value that is the extraction character value that receives the new token value.

- `token-value`:
  the value of the token that is to be inserted into `extraction-char`.

- `token-name`:
  the name of the token. The specified token must be enabled by the extraction definition.

- `extraction-definition`:
  the name of the extraction definition. The definition must exist in the locale that is used. The extraction definition must be the same definition that was used to extract the `extraction-char` value.
**Optional Argument**

**locale**

specifies a character constant, variable, or expression that contains the locale name.

**Default**
The default locale is the first locale in the locale list. If no value is specified, the default locale is used.

**Details**

The DQEXTTOKENPUT function enables you to insert a new value that is associated with a specified token into an extracted value. If a value exists for that token in the input value, the new value is inserted before the existing value. The existing value is retained.

You can specify a variable name for the value of `extraction-char`, and then assign the return value from DQEXTTOKENPUT to the same variable.

**See Also**

**Functions**

- “DQGENDERINFOGET Function” on page 97
- “DQGENDERPARSED Function” on page 98
- “DQMATCHPARSED Function” on page 106
- “DQEXTTOKENGET Function” on page 94

**DQGENDER Function**

Returns a gender determination from the name of an individual.

**Valid in:** DATA step, PROC SQL, and SAS Component Language

**Requirement:** The specified locale must be loaded into memory as part of the locale list.

**Syntax**

`DQGENDER(char, 'gender-analysis-definition' <, 'locale'>)`

**Required Arguments**

`char`

specifies a character constant, variable, or expression that contains the value that is evaluated to determine the gender.

`gender-analysis-definition`

specifies the gender analysis definition, which must exist in the specified locale. The value must be the name of a character variable, in quotation marks. An expression that evaluates to a variable name, or a quoted value is also valid.

You can run the %DQPUTLOC autocall macro to determine which gender definitions are present in your Quality Knowledge Base.
Optional Argument

locale

specifies a character constant, variable, or expression that contains the locale name.

Default The default locale is the first locale in the locale list. If no value is specified, the default locale is used.

Details

The DQGENDER function evaluates the name of an individual to determine the gender of that individual. If the evaluation finds substantial clues that indicate gender, the function returns a value that indicates that the gender is female or male. If the evaluation is inconclusive, the function returns a value that indicates that the gender is unknown. The exact return value is determined by the specified gender analysis definition and locale.

Example: DQGENDER Function

The following example returns the value M for the variable GENDER.

gender=dqGender('Mr. John B. Smith', 'Gender', 'ENUSA');

The gender-analysis-definition must exist in the specified locale. Because recent versions of the locales use the variable NAME instead of GENDER, this example could also be coded as follows:

gender=dqGender('Mr. John B. Smith', 'Name', 'ENUSA');

See Also

Functions

- “DQGENDERPARSED Function” on page 98

DQGENDERINFOGET Function

Returns the name of the parse definition that is associated with the specified gender definition.

Valid in: DATA step, PROC SQL, and SAS Component Language

Requirement: The specified locale must be loaded into memory as part of the locale list.

Syntax

DQGENDERINFOGET('gender-analysis-definition' <',locale'>)

Required Argument

gender-analysis-definition

specifies the gender analysis definition that must exist in the specified locale. The value must be the name of a character variable, in quotation marks. Also valid, an expression that evaluates to a variable name, or a quoted value.
**Optional Argument**

**locale**

specifies a character constant, variable, or expression that contains the locale name.

**Default**
The default locale is the first locale in the locale list. If no value is specified, the default locale is used.

---

**Example: DQGENDERINFOGET Function**

The following example writes the parse definition that is associated with GENDER to the SAS log. The parse definition that is returned is then used to display the names of the tokens that are enabled for that parse definition. The tokens are then used to construct a parsed value and write the results of the gender to the log.

```sas
/* display the parse definition associated with the */
/* GENDER definition and display the tokens in that */
/* parse definition. */
data _null_;  
parseDefn=dqGenderInfoGet('Gender', 'ENUSA');  
tokens=dqParseInfoGet(parseDefn, 'ENUSA');  
put parseDefn= / tokens=;  
run;

/* build a parsed value from two tokens and display */
/* in the log the gender determination for that value. */
data _null_;  
length parsedValue $ 200 gender $ 1;  
parsedValue=dqParseTokenPut(parsedValue, 'Sandi', 'Given Name', 'Name');  
parsedValue=dqParseTokenPut(parsedValue, 'Baker', 'Family Name', 'Name');  
gender=dqGenderParsed(parsedValue, 'Gender');  
put gender=;  
run;
```

**See Also**

**Functions**

- “DQGENDER Function” on page 96
- “DQGENDERPARSED Function” on page 98
- “DQPARSE Function” on page 108
- “DQPARSETOKENPUT Function” on page 114

---

**DQGENDERPARSED Function**

Returns the gender of an individual.

**Valid in:** DATA step, PROC SQL, and SAS Component Language

**Requirement:** The specified locale must be loaded into memory as part of the locale list.
Syntax

DQGENDERPARSED(parsed-char, 'gender-analysis-definition' <locale>)

Required Arguments

parsed-char
the value that is analyzed to determine the gender of an individual. The value must
be the name of a character variable, or a character value in quotation marks. Also
valid, an expression that evaluates to a variable name or quoted value.

gender-analysis-definition
specifies the name of the gender analysis definition. The analysis definition must
exist in the locale that is used.

Optional Argument

locale
specifies a character constant, variable, or expression that contains the locale name.

Default The default locale is the first locale in the locale list. If no value is
specified, the default locale is used.

Details

The DQGENDERPARSED function returns a gender determination from a parsed
character value that contains the name of an individual. If the analysis finds substantial
clues that indicate the gender of the individual, the function returns a value that indicates
that the gender is female or male. If the analysis is inconclusive, the function returns a
value that indicates that the gender is unknown. The specific return value depends on the
specified gender analysis definition and locale.

See Also

Functions

• “DQGENDER Function” on page 96
• “DQGENDERINFOGET Function” on page 97

DQIDENTIFY Function

Returns a category name from a character value.

Valid in: DATA step, PROC SQL, and SAS Component Language

Requirement: The specified locale must be loaded into memory as part of the locale list.

Syntax

DQIDENTIFY(char, 'identification-definition' <locale>)
Required Arguments

dqid specifies a character constant, variable, or expression that contains the value that is analyzed to determine that category of the content.

locale-guess-definition specifies a character constant, variable, or expression that contains the locale name.

The default locale is the first locale in the locale list. If no value is specified, the default locale is used.

Optional Argument

locale specifies a character constant, variable, or expression that contains the locale name.

Details

The DQIDENTIFY function returns a value that indicates the category of the content in an input character value. The available categories and return values depend on your choice of identification definition and locale.

Example: DQIDENTIFY Function

The following example determines whether a character value represents an individual or an organization.

```
dqid=dqIdentify('LL Bean','Individual/Organization','ENUSA');
```

After this function call, the value of DQID is Organization.

DQLOCALEGUESS Function

Returns the name of the locale that is most likely represented by a character value.

Valid in: DATA step, PROC SQL, and SAS Component Language

Syntax

DQLOCALEGUESS(source-string, 'locale-guess-definition')

Required Arguments

source-string specifies a character constant, variable, or expression that contains the value that is analyzed to determine the locale, according to the specified locale guess definition.

locale-guess-definition specifies a character constant, variable, or expression that contains the name of the locale guess definition that is used to evaluate the source string.
Details

The DQLOCALEGUESS function applies the source string to the specified locale guess definition in each of the locales that are listed in the DQLOCALE= system option. Each locale receives a confidence score. The DQLOCALEGUESS function returns the abbreviated name of the locale that receives the highest confidence score.

Note: At least one locale guess definition must exist in each of the locales that are listed in the DQLOCALE= system option. If a locale does not contain a locale guess definition, then the DQLOCALEGUESS function fails to execute.

If two or more locales receive the highest confidence score, then the DQLOCALEGUESS function selects a return value as follows. The return value is the abbreviated name of the highest scoring locale that appears last in the value of the DQLOCALE= system option.

If all locales receive a confidence score of zero, then the DQLOCALEGUESS function returns a value that depends on a setting in the locale guess definitions. The setting is Select the last locale given if no score can be computed. The DQLOCALEGUESS function looks for the setting in each locale, starting with the last locale in DQLOCALE= and working toward the list. The first locale that contains a locale guess definition that enables the setting becomes the name that is returned by the function. If none of the locale guess definitions enable the setting, then the return value of the function is indeterminate.

Examples

Example 1: DQLOCALEGUESS Function
The following example returns the name of a locale as the value of the LOCALE variable.

```plaintext
locale = dqLocaleGuess('101 N. Main Street', 'Address');
```

Example 2: Multiple Highest Confidence Scores
Assume that the confidence scores in the following example are ENUSA=750, ENGBR=750, and ENHGK=250. The value of LOCALE is ENGBR.

```plaintext
options dqlocale=(ENUSA ENGBR ENHGK);
data _null_; locale = dqLocaleGuess(input, 'Country'); run;
```

Example 3: All Locales Have Zero Confidence Scores
In the following example, assume that the setting Select the last locale given if no score can be computed is applied to the locale guess definition in the ENUSA and ENGBR locales. Also assume that the value is not set in the locale guess definition in the ENHGK locale. The value of LOCALE is ENGBR.

```plaintext
options dqlocale=(ENUSA ENGBR ENHGK);
data _null_; locale = dqLocaleGuess(input, 'Country'); run;
```

See Also

“DQLOCALEINFOGET Function” on page 102
DQLOCALEINFOGET Function

Returns information about locales.

Valid in: DATA step, PROC SQL, and SCL

Syntax

DQLOCALEINFOGET(<'info-type'>)

Optional Argument

info-type
the value that is analyzed to determine the locales that are currently loaded into
memory. If no parameter is specified, the default LOADED is used. The only valid
value is LOADED.

Details

The DQLOCALEINFOGET function returns a comma-delimited list of locale names.
The ordered list contains the names of the locales that are currently loaded into memory.
These locales are available for use in data cleansing.

Example: DQLOCALEINFOGET Function

The following example returns the locales that are currently loaded into memory.

    loadedLocales=dqLocaleInfoGet('loaded');
    put loadedLocales;

If the locales ENUSA and ENGBR are loaded in that order, ENUSA,ENGBR is returned.
ENUSA is the default locale.

See Also

Function and autocall macro

- “DQLOCALEINFOLIST Function” on page 102
- “%DQPUTLOC AUTOCALL Macro” on page 70

DQLOCALEINFOLIST Function

Returns the names of the definitions in a locale and a count of those definitions.

Valid in: DATA step, PROC SQL, and SAS Component Language

Requirement: The specified locale must be loaded into memory as part of the locale list.

Syntax

DQLOCALEINFOLIST(<'definition-type'>, <'locale'>)
**Required Arguments**

*definition-type*

specifies the value that is analyzed to determine the names and count of the definition type. The definition type must exist in the specified locale.

Definition types are as follows:

- ALL
- CASE
- EXTRACTION
- GENDER
- GUESS
- IDENTIFICATION
- MATCH
- PARSE
- PATTERN
- STANDARDIZATION

*locale*

specifies a character constant, variable, or expression that contains the locale name. If no value is specified, the default locale is used.

**Default**

The default locale is the first locale in the locale list.

**Interaction**

The DQLOCALEINFOLIST function writes the names of the type-definitions to the SAS log. The return value of the function is the total number of type-definitions.

**See**

- “%DQPUTLOC AUTOCALL Macro” on page 70 for additional information.
- “DQLOCALEINFOGET Function” on page 102 for additional information.

**Example: DQLOCALEINFOLIST Function**

The following example writes a list of the definition names and count in the first locale in the locale list to the SAS log.

```sas
num=dqLocaleInfoList('all');
```

The following example writes a list of parse definitions in the DEDEU locale to the SAS log.

```sas
num=dqLocaleInfoList('parse', 'DEDEU');
```
DQMATCH Function

Returns a match code from a character value.

Valid in: DATA step, PROC SQL, and SAS Component Language

Requirement: The specified locale must be loaded into memory as part of the locale list.

Syntax

DQMATCH(char, 'match-definition '<,'sensitivity'> '<locale'>)

Required Arguments

char
specifies a character constant, variable, or expression that contains the value for which a match code is created, according to the specified match definition.

match-definition
specifies the name of the match definition. The definition must exist in the locale that is used.

Optional Arguments

sensitivity
specifies an integer value that determines the amount of information in the returned match code. Valid values range from 50 to 95. The default value is 85. A higher sensitivity value includes more information in the match code. In general, higher sensitivity values result in a greater number of clusters, with fewer members per cluster, because matches require greater similarity between input values.

locale
specifies a character constant, variable, or expression that contains the locale name.

Default The default locale is the first locale in the locale list. If no value is specified, the default locale is used.

Details

The DQMATCH function parses the input character value and creates a match code. The match code represents a condensed version of the character value. The amount of information in the match code is determined by the sensitivity level. For higher sensitivities, two values must be very similar to produce the same match codes. At lower sensitivities, two values produce the same match codes despite their dissimilarities.

Example: DQMATCH Function

The following example returns a match code that contains the maximum amount of information about the input value.

mcName=dqMatch('Dr. Jim Goodnight', 'NAME', 95, 'ENUSA');
See Also

Functions

- Chapter 8, “DQMATCH Procedure,” on page 37

**DQMATCHINFOGET Function**

Returns the name of the parse definition that is associated with a match definition.

**Valid in:** DATA step, PROC SQL, and SAS Component Language

**Requirement:** The specified locale must be loaded into memory as part of the locale list.

**Syntax**

DQMATCHINFOGET('match-definition' '<locale>')</n

**Required Argument**

*match-definition*

the name of the match definition. The definition must exist in the locale that is used.

**Optional Argument**

*locale*

specifies a character constant, variable, or expression that contains the locale name.

**Default** The default locale is the first locale in the locale list. If no value is specified, the default locale is used.

**Details**

The DQMATCHINFOGET function returns the name of the parse definition that is associated with the specified match definition. Obtaining the name of that parse definition enables you to create parsed character values with the DQPARSE or DQPARSETOKENPUT functions.

If the specified match definition does not have an associated parse definition, the DQMATCHINFOGET function will return a zero-length character variable. The returned missing value indicates that the function has run successfully but has not found an associated parse definition.

**Example: DQMATCHINFOGET Function**

The following example displays the name of the parse definition that is associated with the NAME match definition in the ENUSA locale. That parse definition is then used to display the tokens that are enabled for that parse definition. The tokens are then used to construct a parsed value, create and return a match code, and display the match code.

```plaintext
data _null_;  
parseDefn=dqMatchInfoGet('Name', 'ENUSA');  
tokens=dqParseInfoGet(parseDefn);  
put parseDefn= / tokens=;  
```
DQMATCHPARSED Function

Returns a match code from a parsed character value.

**Valid in:** DATA step, PROC SQL, and SAS Component Language

**Requirement:** If specified, the locale must be loaded into memory as part of the locale list.

**Syntax**

\[
\text{DQMATCHPARSED}(\text{parsed-char}, \text{match-definition} <,\text{sensitivity}> <,\text{locale}>)
\]

**Required Arguments**

parsed-char

specifies a character constant, variable, or expression that contains the value that is the name of the parsed-definition that is associated with the match definition.

To determine the name of the associated parse definition, use the DQMATCHINFOGET function. To determine the tokens that are enabled by that parse definition, use the DQPARSEINFOGET function.

match-definition

specifies the name of the match definition. The definition must exist in the locale that is used.

**Optional Arguments**

sensitivity

specifies an integer value that determines the amount of information in the returned match code. Valid values range from 50 to 95. The default value is 85. A higher sensitivity value inserts more information in the match code. In general, higher sensitivity values result in a greater number of clusters, with fewer members per cluster. Input values must be more similar to receive the same match codes.

locale

specifies a character constant, variable, or expression that contains the locale name.

**Default** The default locale is the first locale in the locale list. If no value is specified, the default locale is used.

**Example: DQMATCHPARSED Function**

The following example returns a match code for the parsed name of an individual. The amount of information in the match code is high.

```sas
data _null_;
length parsedValue $ 200 matchCode $ 15;
parsedValue=dqParseTokenPut(parsedValue, 'Joel', 'Given Name', 'Name');
parsedValue=dqParseTokenPut(parsedValue, 'Alston', 'Family Name', 'Name');
matchCode=dqMatchParsed(parsedValue, 'Name');
put matchCode=;
run;
```
length nameIndividual matchCode $ 20 parsedName $ 200;
nameIndividual='Susan B. Anthony';
parsedName=dqParse(nameIndividual, 'name', 'enusa');
matchCode=dqMatchParsed(parsedName, 'name', 90, 'enusa');
run;

See Also

Functions

• Chapter 8, “DQMATCH Procedure,” on page 37
• “DQMATCHINFOGET Function” on page 105
• “DQPARSEINFOGET Function” on page 110
• “DQTOKEN Function” on page 126

DQOPTSURFACE Function

Reveals or hides non-surfaced definitions.

Valid in: DATA step, PROC SQL, and SAS Component Language

Syntax

DQOPTSURFACE('surface-definition')

Required Argument

surface-definition

specifies the policy for the surface definitions.

Details

The DQOPTSURFACE function specifies whether the non-surfaced definitions are
revealed or hidden. By default, non-surfaced definitions are hidden. Valid input values
are as follows:

YES

reveals the non-surfaced definitions.

NO

hides the non-surfaced definitions.

The DQOPTSURFACE function returns the previous value of the surface definition
policy.

Example: DQOPTSURFACE Function

The following example specifies that non-surfaced definitions are revealed. The
character value oldDEFAULT contains the value of the previous setting.

oldDefault=DQOPTSURFACE('YES');
**DQPARSE Function**

Returns a parsed character value.

- **Valid in:** DATA step, PROC SQL, and SAS Component Language
- **Restriction:** Always use the DQPARESETOKENGET function to extract tokens from parsed values. To extract tokens from values that do not contain delimiters, use the DQTOKEN function.
- **Requirement:** If specified, the locale must be loaded into memory as part of the locale list.

**Syntax**

\[
\text{DQPARSE}\left('\text{parse-string}', 'parse-definition' <',locale'>\right)
\]

**Required Arguments**

- **parse-string**
  - the value that is parsed according to the specified parse definition. The value must be the name of a character variable, or a character value in quotation marks. Also valid, an expression that evaluates to a variable name or quoted value.

- **parse-definition**
  - the name of the parse definition. The definition must exist in the locale that is used.

**Optional Argument**

- **locale**
  - specifies a character constant, variable, or expression that contains the locale name.
  - **Default** The first locale in the locale list. If no value is specified, the default locale is used.

**Details**

The DQPARSE function returns a parsed character value. The return value contains delimiters that identify the elements in the value that correspond to the tokens that are enabled by the parse definition. The delimiters in the value allow functions such as DQPARESETOKENGET to access the elements in the value based on specified token names.

**Example: DQPARSE Function**

The following example parses the name of an individual. Then the DQPARESETOKENGET function returns the values of two of the tokens.

\[
\begin{align*}
\text{parsedValue} &= \text{dqParse}('\text{Mrs. Sallie Mae Pravlik}', 'NAME', 'ENUSA'); \\
\text{prefix} &= \text{dqParseTokenGet} (\text{parsedValue}, 'Name Prefix', 'NAME', 'ENUSA'); \\
\text{given} &= \text{dqParseTokenGet} (\text{parsedValue}, 'Given Name', 'NAME', 'ENUSA');
\end{align*}
\]

After these function calls, the value of PREFIX is Mrs. and the value of GIVEN is Sallie.
See Also

• “DQPARSEINFOGET Function” on page 110
• “DQTOKEN Function” on page 126

DQPARSE CALL Routine

Returns a parsed character value and a status flag.

Valid in: DATA step, PROC SQL, and SAS Component Language
Restriction: Always use the DQPARSETOKENGET function to extract tokens from parsed values. To extract tokens from values that do not contain delimiters, use the DQTOKEN function.
Requirement: If specified, the locale must be loaded into memory as part of the locale list.

Syntax

CALL DQPARSE(parse-string, parse-definition, 'parse-result, parse-return-code <, 'locale '>);

Required Arguments

parse-string
the input value that is parsed according to the specified parse definition. The value must be the name of a character variable, or a character value in quotation marks. Also valid, an expression that evaluates to a variable name or quoted value.

parse-definition
the name of the parse definition. The definition must exist in the locale that is used.

parse-result
an output character variable that receives the result of the parse operation.

parse-return-code
an output numeric variable that returns 1 when the parse operation is successful. Otherwise, this variable receives a 0.

Optional Argument

locale
specifies a character constant, variable, or expression that contains the locale name.

Default The default locale is the first locale in the locale list. If no value is specified, the default locale is used.

Details

The DQPARSE CALL routine returns a parsed character value and a return code into separate variables. The parsed character value contains delimiters that identify the elements in the value that correspond to the tokens that are enabled by the parse definition. The delimiters in the value allow functions such as DQPARSETOKENGET to access the elements in the value based on specified token names.
Example: DQPARSE CALL Routine

The following example parses the name of an individual.

```sas
data a;
  length parsename $ 40;
  call dqparse (name, 'Name', parsename, solution);
  if solution= 1 then
    put 'found solution';
  else
    put 'no solution';
run;
```

See Also

Functions

- “DQPARSEINFOGET Function” on page 110
- “DQTOKEN Function” on page 126

DQPARSEINFOGET Function

Returns the token names in a parse definition.

Valid in: DATA step, PROC SQL, and SAS Component Language

Required Argument

`parse-definition` specifies the name of the parse definition. The definition must exist in the locale that is used.

Optional Argument

`locale` specifies a character constant, variable, or expression that contains the locale name.

Default The default locale is the first locale in the locale list. If no value is specified, the default locale is used.

Details

The DQPARSEINFOGET function returns the names of the tokens that can be inserted into character values using the DQPARSETOKENPUT function.
Example: DQPARSEINFOGET Function

The following example returns the token names for the parse definition email in the locale ENUSA and displays the token names in the SAS log.

```sas
tokenNames=dqParseInfoGet('e-mail','ENUSA');
put tokenNames;
```

After this function call, the value of TOKENNAMES is Mailbox, Sub-Domain, Top-Level Domain, which are the names of the three tokens in this parse definition.

See Also

Functions

- “DQPARSETOKENGET Function” on page 113
- “DQPARSETOKENPUT Function” on page 114
- “DQTOKEN Function” on page 126

DQPARSEINPUTLEN Function

Sets the default length of parsed input, and returns a string indicating its previous value.

Valid in: DATA step, PROC SQL, and SAS Component Language

Syntax

```
DQPARSEINPUTLEN('input-length')
```

Required Argument

`input-length`

specifies the input length for parsing functions. DQPARSEINPUTLEN returns the previous value of the input length.

Details

The DQPARSEINPUTLEN function specifies the input length anticipated by parsing functions. If REMOVE is specified, the override value is removed and the input limit is set to the default value. Valid values for the input length are as follows:

- SHORT
- LONG
- AUTO
- REMOVE

The DQPARSEINPUTLEN function returns a value that indicates the previous value of the input length. If the value NOTSET is returned, the override value is not set. Possible values for the previous input length are as follows:

- SHORT
- LONG
- AUTO
Example: **DQPARSEINPUTLEN Function**

The following example sets the default input length to SHORT. The previous value of the parse input length is returned as the value of `oldDEFAULT`.

```plaintext
oldDefault= dqParseInfPutLen('short');
```

---

**DQPARSERESLIMIT Function**

Sets a limit on resources consumed during parsing.

**Valid in:** DATA step, PROC SQL, and SAS Component Language

**Syntax**

```plaintext
DQPARSERESLIMIT('resource-limit')
```

**Required Argument**

`resource limit`

specifies the resource limit a parsing operation is allowed to consume.

**Details**

The DQPARSERESLIMIT function sets the level of resource consumption in force during parsing operations. If REMOVE is specified, the override value is removed and the resource limit is set to the default value. Valid values are as follows:

- VERYLOW
- LOW
- MEDIUM
- HIGH
- VERYHIGH
- INTENSIVE
- REMOVE

The DQPARSERESLIMIT function returns a value that indicates the previous value of the resource limit. If the value NOTSET is returned, the override value is not set. Possible return values are as follows:

- VERYLOW
- LOW
- MEDIUM
- HIGH
- VERYHIGH
- INTENSIVE
- NOTSET
Example: DQPARSERESLIMIT Function

The following example sets the default resource limit to INTENSIVE. The value of oldDEFAULT is the previous value of the resource limit.

```python
oldDefault=DQPARSERESLIMIT('intensive');
```

DQPARSESCORDEPTH Function

Specifies how deeply to search for the best parsing score.

**Valid in:** DATA step, PROC SQL, and SAS Component Language

**Syntax**

```python
DQPARSESCORDEPTH(level)
```

**Required Argument**

`level`

the maximum depth permitted during scoring.

**Details**

The DQPARSESCORDEPTH function sets the level of how deeply to search for the best parsing score. LEVEL must be in the range from five to ten inclusive, or zero. If at least one of the conditions is true, DQPARSESCORDEPTH overrides the default scoring depth value. If zero is returned, there is no override value in force.

The DQPARSESCORDEPTH function returns the previous value of the override solutions depth.

Example: DQPARSESCORDEPTH Function

The following example sets DQPARSESCORDEPTH to eight. The numeric variable oldDEFAULT contains the scoring depth previously in force.

```python
oldDefault=DQPARSESCORDEPTH(8);
```

DQPARESETOKENGET Function

Returns a token from a parsed character value.

**Valid in:** DATA step, PROC SQL, and SAS Component Language

**Restriction:** Do not attempt to extract tokens from parsed values using any means other than the DQPARESETOKENGET function.

**Requirement:** If specified, the locale must be loaded into memory as part of the locale list.

**Syntax**

```python
DQPARESETOKENGET(parsed-char, 'token', 'parse-definition' <,'locale'>)
```
**Required Arguments**

**parsed-char**

specifies a character constant, variable, or expression that contains the value that is the parsed character value from which the value of the specified token is returned.

To determine how the parse definition inserts delimiters, use the DQPARSEINFOGET function.

**token**

the name of the token that is returned from the parsed value. The token must be enabled by the specified parse definition

**parse-definition**

the name of the parse definition. The definition must exist in the locale that is used. The parse definition must be the same as the parse definition that originally parsed the PARSED-CHAR value.

**Optional Argument**

**locale**

specifies a character constant, variable, or expression that contains the locale name.

**Default**

The default locale is the first locale in the locale list. If no value is specified, the default locale is used.

**Details**

The DQPARSETOKENGET function returns the value of the specified token from a previously parsed character value.

**Example: DQPARSETOKENGET Function**

The following example parses a character value with the DQPARSE function and extracts two of the tokens with the DQPARSETOKENGET function.

```qsh
parsedValue=dqParse('Mrs. Sallie Mae Pravlik', 'NAME', 'ENUSA');
prefix=dqParseTokenGet(parsedValue, 'Name Prefix', 'NAME', 'ENUSA');
given=dqParseTokenGet(parsedValue, 'Given Name', 'NAME', 'ENUSA');
```

After these function calls, the value of **prefix** is **Mrs.** and the value of **given** is **Sallie**.

**See Also**

**Functions**

- “DQPARSE Function” on page 108
- “DQPARSEINFOGET Function” on page 110
- “DQTOKEN Function” on page 126

**DQPARSETOKENPUT Function**

Inserts a token into a parsed character value and returns the updated parsed character value.
Valid in: DATA step and SAS Component Language
Requirement: If specified, the locale must be loaded into memory as part of the locale list.

Syntax

DQPARSETOKENPUT('parsed-char', 'token-value', 'token-name', 'parse-definition' <',locale'>)

Required Arguments

parsed-char
specifies a character constant, variable, or expression that contains the value that is the parsed character value that receives the new token value.

token-value
the value of the token that is to be inserted into parsed-char.

token-name
the name of the token. The specified token must be enabled by the parse definition.

parse-definition
the name of the parse definition. The definition must exist in the locale that is used. The parse definition must be the same definition that was used to parse the parsed-char value.

Optional Argument

locale
specifies a character constant, variable, or expression that contains the locale name.

Default The default locale is the first locale in the locale list. If no value is specified, the default locale is used.

Details

The DQPARSETOKENPUT function enables you to insert a new value that is associated with a specified token into a parsed value. If a value exists for that token in the input value, the new value is inserted before the existing value. The existing value is retained.

You can specify a variable name for the value of parsed-char, and then assign the return value from DQPARSETOKENPUT to the same variable.

See Also

Functions

- “DQGENDERINFOGET Function” on page 97
- “DQGENDERPARSED Function” on page 98
- “DQMATCHPARSED Function” on page 106
- “DQPARSETOKENGET Function” on page 113
DQPATTERN Function
Returns a pattern analysis from an input character value.

Valid in: DATA step and SAS Component Language
Requirement: If specified, the locale must be loaded into memory as part of the locale list.

Syntax

DQPATTERN('char', 'pattern-analysis-definition' <locale>)

Required Arguments

char
specifies a character constant, variable, or expression that contains the value that is the name of the input character value that is analyzed.

pattern-analysis-definition
the name of the pattern analysis definition. The definition must exist in the locale that is used.

Optional Argument

locale
specifies a character constant, variable, or expression that contains the locale name.

Default The default locale is the first locale in the locale list. If no value is specified, the default locale is used.

Details

The DQPATTERN function returns a pattern analysis from an input character value. DQPATTERN identifies words or characters in the input value as numeric, alphabetic, non-alphanumeric, or mixed. The choice of pattern analysis definition determines the nature of the analysis as follows:

* non-alphanumeric, such as punctuation marks or symbols
A alphabetic
M mixture of alphabetic, numeric, and non-alphanumeric
N numeric

Example: DQPATTERN Function

The following example analyzes the words in the input character value. The results are written to the SAS log using the PUT statement.

pattern=dqPattern('WIDGETS 5','32CT','WORD','ENUSA');
put pattern;
The DQPATTERN function returns \( A N^* M \). Using the CHARACTER pattern analysis definition returns \( AAAAAAA N^* NNAA \).

**DQSCHEMEAPPLY Function**

Applies a scheme and returns a transformed value.

**Valid in:**
- DATA step, PROC SQL, and SAS Component Language

**Requirements:**
- If specified, the locale must be loaded into memory as part of the locale list.
- Schemes using SAS format are required in the z/OS operating environment.

**Syntax**

```
DQSCHEMEAPPLY(char, 'scheme', 'scheme-format', '<mode>',<,'scheme-lookup-method'> <,'match-definition'> <,sensitivity> <,locale>)
```

**Required Arguments**

- **char**: specifies a character constant, variable, or expression that contains the value to which the specified scheme is applied.

- **scheme**: identifies the scheme that is applied to the input value. For schemes using SAS format, the `scheme` argument includes both the path and the filename of the SAS data set, in quotation marks.

For schemes using QKB scheme file format, the `scheme` argument is the name of an existing fileref in quotation marks. For all operating environments other than z/OS, the fileref must reference a file specification that includes both the path and the filename that ends in `.sch.qkb`.

**Requirement**
- Lowercase characters are required.

**Note**
In the z/OS operating environment, the normal naming conventions apply for the partitioned data set (PDS) that contains the scheme.

- **scheme-format**: identifies the format of the scheme. The valid values for this argument are defined as follows:
  - **QKB**: indicates that the scheme is stored in QKB scheme file format. This is the default value.
  - **NOQKB**: indicates that the scheme is stored in SAS format.

**See**
- “Creating Schemes” on page 60
**Optional Arguments**

**mode**

specifies how the scheme is to be applied to the values of the input character variable.

If the value of scheme-lookup-method is USE_MATCHDEF, and a value is not specified for mode, the default value of mode (PHRASE) is used.

Valid values for mode are as follows:

**PHRASE**
compares the entire input character value to the entire length of each of the DATA values in the scheme. When the value of the scheme-lookup-method is USE_MATCHDEF, the match code values of the entire input value are compared to the match codes of DATA values in the scheme. A transformation occurs when a match is found between an element in the input value and a DATA value in the scheme.

**ELEMENT**
compares each element in the input character value to each of the DATA values in the scheme. When the value of the scheme-lookup-method is USE_MATCHDEF, the match code of the entire input value is compared to the match codes of the scheme's DATA values. A transformation occurs when a match is found between an element in the input value and a DATA value in the scheme.

**Default**

The mode that is stored in the scheme. If a mode is not stored in the scheme, the default value of PHRASE is used.

**scheme-lookup-method**

specifies one of three mutually exclusive methods of applying the scheme.

**EXACT**
(default value) specifies that the input value is to be compared to the DATA values in the scheme without changing the input value in any way. The transformation value in the scheme is written into the output data set only when the input value exactly matches a DATA value in the scheme. Any adjacent blank spaces in the input value are replaced with single blank spaces before comparison.

**IGNORE_CASE**
specifies that capitalization is to be ignored when the input value is compared to the DATA values in the scheme. Any adjacent blank spaces in the input value are replaced with single blank spaces before comparison.

**USE_MATCHDEF**
specifies that the match code of the input value is to be compared to the match code of the DATA values in the scheme. A transformation occurs when the match codes are identical.

Specify USE_MATCHDEF to enable locale, match-definition, and sensitivity.

**Default**

EXACT

**Restriction**
The arguments locale, match-definition, and sensitivity are valid only when the value of scheme-lookup-method is USE_MATCHDEF.

**match-definition**

the name of the match definition. The definition must exist in the locale that is used to create match codes during the application of the scheme. If USE_MATCHDEF is
specified and a match definition is not stored in the scheme, then a value is required for the match-definition argument.

**Default**
If USE_MATCHDEF is specified and the match-definition argument is not specified, then the default match definition is the one that is stored in the scheme.

**Restriction**
The match-definition argument is valid only when the value of the scheme-lookup-method argument is USE_MATCHDEF.

**See**
“Meta Options” on page 61

**sensitivity**
specifies the amount of information in the match codes that are created during the application of the scheme. With higher sensitivity values, two values must be increasingly similar to create the same match code. At lower sensitivity values, values can receive the same match code despite their dissimilarities.

**Default**
When use_matchdef is specified and the sensitivity argument is not specified, the default sensitivity is the sensitivity value that is stored in the scheme. When USE_MATCHDEF is specified and a sensitivity value is not stored in the scheme, the default sensitivity value is 85.

**Range**
50 to 95

**Restriction**
The sensitivity argument is valid only when the value of the scheme-lookup-method argument is USE_MATCHDEF.

**Note**
To return a count of the number of transformations that take place during a scheme application, use the DQSCHEMEAPPLY CALL routine.

**See**
“Meta Options” on page 61

**locale**
specifies a character constant, variable, or expression that contains the locale name.

**Default**
The default locale is the first locale in the locale list. If no value is specified, the default locale is used.

**Details**
The locale argument is valid only when the value of the scheme-lookup-method argument is USE_MATCHDEF. The DQSCHEMEAPPLY function transforms an input value by applying a scheme. The scheme can be in SAS format or QKB format. To create schemes in SAS format, use the DQSCHEME procedure. To create schemes in QKB format, use the DQSCHEME procedure or use the DataFlux Data Management Studio software.

**Example: DQSCHEMEAPPLY Function**
The following example generates a scheme with the DQSCHEME procedure and then applies that scheme to a data set with the DQSCHEME function. The example assumes that the ENUSA locale has been loaded into memory as part of the locale list.

```/* Create the input data set. */
data suppliers;```
length company $ 50;
   input company $char50.;
datalines;
Ford Motor Company
Walmart Inc.
Federal Reserve Bank
Walmart
Ernest & Young
TRW INC - Space Defense
Wal-Mart Corp.
The Jackson Data Corp.
Ernest & Young
Federal Reserve Bank 12th District
Ernest and Young
Jackson Data Corp.
Farmers Insurance Group
Kaiser Permantente
Ernest and Young LLP
TRW Space & Defense
Ford Motor
Jackson Data Corp
Federal Reserve Bank
Target
;
run;

/* Assign a fileref to the scheme file. */
filename myscheme 'c:\temp\company.sch.qkb';

/* Create the scheme. */
proc dqscheme data=suppliers qkb;
   create matchdef='Organization (Scheme Build)'
      var=company scheme=myscheme
      locale='ENUSA';
run;

/* Apply the scheme and display the results. */
data suppliers;
   set suppliers;
   length outCompany $ 50;
   outCompany=dqSchemeApply(company,'myscheme','qkb','phrase','EXACT');
   put 'Before applying the scheme: ' company /
       'After applying the scheme: ' outCompany;
run;

See Also
Chapter 9, “DQSHEME Procedure,” on page 53

DQSHEMEAPPLY CALL Routine
Applies a scheme and returns a transformed value and a transformation flag.

Valid in:   DATA step and SAS Component Language

Requirements:  If specified, the locale must be loaded into memory as part of the locale list.
Schemes using SAS format are required in the z/OS operating environment.

**Syntax**

```call
call dqschemeapply('char', 'output-variable', 'scheme', 'scheme-format'

  <, 'mode'> <, 'transform-count-variable'> <, 'scheme-lookup-method'>

  <, 'match-definition'> <, 'sensitivity'> <, 'locale'>);
```

**Required Arguments**

- **char**
  - specifies a character constant, variable, or expression that contains the value that is the input value to which the scheme is applied.

- **output-variable**
  - the character variable that receives the transformed input value.

- **scheme**
  - the scheme that is applied to the input value. A SAS format scheme is a filename specification that includes a pathname and the SAS data set name enclosed in quotation marks.
  
  QKB scheme file format scheme includes the name of an existing fileref in quotation marks. For all operating environments other than z/OS, the fileref must reference a file specification that includes both the pathname and the filename that ends in .sch.qkb.

  **Requirement**  Lowercase letters.

  **Note**  In the z/OS operating environment, the normal naming conventions apply for the partitioned data set (PDS) that contains the scheme.

- **scheme-format**
  - identifies the format of the scheme. The valid values for this argument are defined as follows:
    
    - **QKB**
      - indicates that the scheme is stored in QKB scheme file format. This is the default value.
    
    - **NOQKB**
      - indicates that the scheme is stored in SAS format.

  **Default**  QKB

  **See**  “Creating Schemes” on page 60

**Optional Arguments**

- **mode**
  - specifies how the scheme is to be applied to the values of the input character variable. The default value of mode is the mode that is stored in the scheme. If a mode is not stored in the scheme, the default value of mode, is PHRASE.

  If the value of scheme-lookup-method is USE_MATCHDEF, and a value is not specified for mode, the default value of mode (PHRASE) is used.

  Valid values for mode are as follows:
PHRASE
compares the entire input character value to the entire length of each of the DATA values in the scheme. When the value of the scheme-lookup-method is USE_MATCHDEF, the match code values of the entire input value are compared to the match codes of DATA values in the scheme. A transformation occurs when a match is found between an element in the input value and a DATA value in the scheme.

ELEMENT
compares each element in the input character value to each of the DATA values in the scheme. When the value of the scheme-lookup-method is USE_MATCHDEF, the match code of the entire input value is compared to the match codes of the scheme's DATA values. A transformation occurs when a match is found between an element in the input value and a DATA value in the scheme.

transform-count-variable
identifies the numeric variable that receives the returned number of transformations that were performed on the input value.

If the input variable is transformed, then the value is a positive integer that represents the number of elements in the input value that are transformed.

Interactions
If the value of mode is PHRASE and the input value is not transformed, then the value of the transform-count-variable is 0.

If the input variable is transformed, the value of transform-count-variable is 1.

If the value of the mode is ELEMENT and the input value is not transformed, then the value of the transform-count-variable is 0.

The transformation count might appear to be inaccurate if the transformation value in the scheme is the same as the input value (or any element in the input value).

scheme-lookup-method
specifies one of three mutually exclusive methods of applying the scheme. Valid values for scheme-lookup-method are as follows:

EXACT
(default value) specifies that the input value is to be compared to the DATA values in the scheme without changing the input value in any way. The transformation value in the scheme is written into the output data set only when the input value exactly matches a DATA value in the scheme. Any adjacent blank spaces in the input value are replaced with single blank spaces before comparison.

IGNORE_CASE
specifies that capitalization is to be ignored when the input value is compared to the DATA values in the scheme. Any adjacent blank spaces in the input value are replaced with single blank spaces before comparison.

USE_MATCHDEF
specifies that the match code of the input value is to be compared to the match code of the DATA values in the scheme. A transformation occurs when the two match codes are identical.

Specifying USE_MATCHDEF enables you to modify the values of locale, match-definition, and sensitivity.
Note: The locale, match-definition, and sensitivity values are valid only when the value of the scheme-lookup-method is USE_MATCHDEF.

**match-definition**

the name of the match definition. The definition must exist in the locale that is used to create match codes during the application of the scheme.

**Interactions**

If USE_MATCHDEF is specified and match-definition is not specified, the default match definition is stored in the scheme.

The match-definition value is valid only when the value of the scheme-lookup-method is USE_MATCHDEF.

If USE_MATCHDEF is specified and a match-definition is not stored in the scheme, then a value is required for match-definition.

**sensitivity**

specifies the amount of information in the match codes that are created during the application of the scheme. With higher sensitivity values, two values must be increasingly similar to create the same match code. At lower sensitivity values, two values receive the same match code despite their dissimilarities.

| Default | 85 |
| Range   | 50 to 95 |

**Interactions**

*Sensitivity* is valid only when the value of the scheme-lookup-method is USE_MATCHDEF.

If sensitivity is not in the scheme and USE_MATCHDEF is specified, the sensitivity value that is stored in the scheme is used.

**locale**

specifies a character constant, variable, or expression that contains the locale name.

**Default**

The default locale is the first locale in the locale list. If no value is specified, the default locale is used.

**Note**

The locale is valid only when the value of the scheme-lookup-method is USE_MATCHDEF.

**Details**

The DQSCHEMEAPPLY CALL routine transforms an input value by applying a scheme. The scheme can be in SAS format or QKB format. To create schemes in SAS format, use the DQSCHEME procedure. To create schemes in QKB format, use the DataFlux Data Management Studio software.

**Example: DQSCHEMEAPPLY CALL Routine**

The following example generates a scheme using QKB scheme file format with the DQSCHEME procedure and then applies that scheme to a data set with the DQSCHEMEAPPLY CALL routine. The example assumes that ENUSA has been loaded into memory as the default locale.

```sas
/* Create the input data set. */
data suppliers;
    length company $ 50;
```
input company $char50.;
datalines;
Ford Motor Company
Walmart Inc.
Federal Reserve Bank
Walmart
Ernest & Young
TRW INC - Space Defense
Wal-Mart Corp.
The Jackson Data Corp.
Ernest & Young
Federal Reserve Bank 12th District
Ernest and Young
Jackson Data Corp.
Farmers Insurance Group
Kaiser Permantente
Ernest and Young LLP
TRW Space & Defense
Ford Motor
Jackson Data Corp
Federal Reserve Bank
Target
;
run;
/* Create the scheme. */
proc dqscheme data=suppliers nogkb;
create matchdef='Organization (Scheme Build)' var=company scheme=work.myscheme locale='ENUSA';
run;
/* Print the scheme. */
proc print data=work.myscheme;
title 'Organization Scheme';
run;
/* Apply the scheme and display the results. */
data suppliers;
set suppliers;
length outCompany $ 50;
call dqSchemeApply(company, outCompany, 'work.myscheme', 'nogkb', 'phrase', numTrans);
put 'Before applying the scheme: ' company /
'After applying the scheme: ' outCompany /
'Transformation count: ' numTrans /;
run;

The value of the NUMTRANS variable is 0 if the organization name is not transformed. The value is 1 if the organization name is transformed. In the following example, a transformation count of 1 is shown in instances, when no transformation appears to have been made. This is shown in the PROC PRINT output.

Before applying the scheme: Jackson Data Corp
After applying the scheme: Jackson Data Corp
Transformation count: 1

Instances such as these are not errors. In these cases the transformation value is the same as the input value.
DQSTANDARDIZE Function

Returns a character value after standardizing casing, spacing, and format, and then applies a common representation to certain words and abbreviations.

**Valid in:** DATA step, PROC SQL, and SAS Component Language

**Requirement:** If specified, the locale must be loaded into memory as part of the locale list.

**Syntax**

DQSTANDARDIZE(char, 'standardization-definition' <, locale>)

**Required Arguments**

`char` specifies a character constant, variable, or expression that contains the value that is standardized according to the specified standardization definition.

`standardization-definition` specifies the name of the standardization definition. The definition must exist in the locale that is used.

**Optional Argument**

`locale` specifies a character constant, variable, or expression that contains the locale name.

**Default** The default locale is the first locale in the locale list. If no value is specified, the default locale is used.

**Details**

In the locales, standardization definitions are provided for character content such as dates, names, and ZIP codes. The available standardization definitions vary from one locale to the next.

The return value is provided in the appropriate case, with insignificant blank spaces and punctuation removed. The standardization definition that was specified in the DQSTANDARDIZE function might standardize certain words and abbreviations. The order of the elements in the return value might differ from the order of the elements in the input character value.

**Example: DQSTANDARDIZE Function**

The following example standardizes four names using the NAME standardization definition from the ENUSA locale. The following example assumes that the ENUSA locale has been loaded into memory as part of the locale list.

```sas
data _null_
length name stdName $ 50;
input name $char50.;
```
stdName=dqStandardize(name, 'Name');
put 'Name:' @10 name / 
   'StdName:' @10 stdName /;
datalines;
HOUSE, KEN
House, Kenneth
House, Mr. Ken W.
MR. KEN W. HOUSE
;
run;

After this function call, the SAS log displays the following information:

Name:   HOUSE, KEN
StdName: Ken House
Name:   House, Kenneth
StdName: Kenneth House
Name:   House, Mr. Ken W.
StdName: Mr Ken W House
Name:   MR. KEN W. HOUSE
StdName: Mr Ken W House

---

**DQTOKEN Function**

Returns a token from a character value.

**Valid in:**  DATA step, PROC SQL, and SAS Component Language

**Requirement:**  If specified, the locale must be loaded into memory as part of the locale list.

---

**Syntax**

**DQTOKEN(char, 'token', 'parse-definition' <, locale>)**

---

**Required Arguments**

**char**

specifies a character constant, variable, or expression that contains the value that is the value from which the specified token is returned, according to the specified parse definition.

**token**

identifies the token that is returned.

**parse-definition**

the name of the parse definition. The definition must exist in the locale that is used.

---

**Optional Argument**

**locale**

specifies a character constant, variable, or expression that contains the locale name.

**Default**  The default locale is the first locale in the locale list. If no value is specified, the default locale is used.
Details

Use the DQTOKEN function to parse a value and return one token. If the DQTOKEN function does not find a value for that token, the return value for that token is blank.

To return more than one token from a parsed value, use the functions DQPARSE and DQPARSETOKENGET.

Example: DQTOKEN Function

The following example parses a single token from a character value:

```latex
prefix=dqToken('Mrs. Sallie Mae Pravlik','Name Prefix','Name','ENUSA');
```

After the DQTOKEN call, the value for the PREFIX variable is Mrs.

See Also

Functions

- “DQPARSE Function” on page 108
- “DQPARSETOKENGET Function” on page 113

DQVER Function

Returns the version of the SAS Data Quality engine.

Valid in: DATA step, PROC SQL, and SAS Component Language

Syntax

DQVER()

Details

The DQVER function takes no arguments and returns the version number of the SAS Data Quality engine that has been loaded with SAS Data Quality Server.

Example: DQVER Function

The following example returns the SAS Data Quality engine version.

```latex
version=DQVER();
```

DQVERQKB Function

Returns the version of the currently loaded QKB.

Valid in: DATA step, PROC SQL, and SAS Component Language
Syntax

DQVERQKB ()

Details

The DQVERQKB function takes no arguments and returns a five-character string that contains the version of the currently loaded QKB. If the version cannot be determined (as with QKB versions before 2005A), the value UNKNW is returned.

Example: DQVERQKB Function

The following example returns the version of the currently loaded QKB.

```
version= DQVERQKB ();
```
Chapter 12
SAS Data Quality Server System Options

The SAS Data Quality Server system options DQLOCALE= and DQSETUPLOC= must be asserted before you run data cleansing programs. The DQOPTIONS= system option is used at SAS invocation to set data quality parameters.

To specify values for the DQLOCALE= and DQSETUPLOC= system options, use the “%DQLOAD AUTOCALL Macro” on page 69.

CAUTION:
It is not recommended that you specify these system options by any means other than invoking the %DQLOAD AUTOCALL macro. Failure to use %DQLOAD or misapplied use of default settings for these system options can result in data that is cleansed with inappropriate locales.

System Options:

- DQLOCALE must be run before running data cleansing programs. See “DQLOCALE= System Option” on page 130 for additional information.

- The DQOPTIONS= system option enables you to optimize your SAS session for data quality. The value of the system option is a set of option-value pairs that you specify on the SAS start-up command or in the SAS configuration file. The data quality system options can be referenced by the OPTIONS procedure by specifying GROUP=DATAQUALITY. See: “DQOPTIONS= System Option” on page 131.

- DQSETUPLOC must be run before running data cleansing programs. See “DQSETUPLOC= System Option” on page 132 for additional information.
Dictionary

DQLOCALE= System Option

Specifies an ordered list of locales.

Valid in:  Configuration file, SAS invocation, OPTIONS statement, SAS System Options window

Category:  Input Control: Data Quality

PROC OPTIONS GROUP=

Requirement:  You must specify at least one locale.

Syntax

DQLOCALE=(locale-1 <, locale-2, ...>)

Action

locale-1 <, locale-2, ...>

specifies an ordered list of locales. The list determines how the data is cleansed. Locales are applied to the data in the order in which they are specified. All locales in the list must exist in the Quality Knowledge Base.

Details

The DQLOCALE= system option identifies the locales that are referenced during data cleansing. The order of the locales in the list affects the locale matching scheme of the DQMATCH procedure.

Unlike other system options, the value of the DQLOCALE= system option must be loaded into memory. Normally, system option values go into the system options table only. Because the locales that are specified with this option must also be loaded into memory, always set the value of this system option by invoking the AUTOCALL macro %DQLOAD. This macro takes as its arguments the values for the DQLOCALE= and DQSETUPLOC= system options.

CAUTION:

It is recommended that you invoke the AUTOCALL macro %DQLOAD at the beginning of each data cleansing program or session. Failure to do so might generate unintended output.

SAS specifies no default value for the DQLOCALE= system option. It is recommended that you not use an AUTOEXEC to load default locales when you invoke SAS. Loading default locales can enable you to apply the wrong locales to your data, which generates unintended output. Loading default locales also wastes resources when you are not cleansing data. Instead of loading default locales, invoke the %DQLOAD macro at the beginning of each data cleansing program or session. See “%DQLOAD AUTOCALL Macro” on page 69 for additional information.
**DQOPTIONS= System Option**

Specifies SAS session parameters for data quality programs.

**Valid in:** Configuration file, SAS invocation  
**Category:** Environment Control: Initialization and Operation  
**PROC OPTIONS GROUP=** EXECMODES  
**Restriction:** You cannot create or apply schemes in QKB scheme file format in z/OS.

### Syntax

```
DQOPTIONS=(DQSRVPROTOCOL=WIRELINE | SOAP <TRANSCODE=IGNORE | WARN><IWA=NO | YES>)
```

### Required Argument

**DQSRVPROTOCOL=WIRELINE | SOAP**  
Designates the SAS Data Quality Server protocol. In operating environments other than z/OS, the default SOAP protocol is recommended.  
- **SOAP** specifies to use the Simple Object Access Protocol (SOAP).  
- **WIRELINE** specifies the Wireline protocol in the z/OS operating environment. The Wireline protocol improves data transfer performance in z/OS. In the SAS Data Quality Server software, z/OS support encompasses the DMSRVDATASVC procedure and all functions.  
  
**Requirement** The Wireline protocol must be specified in the z/OS operating environment.

### Optional Arguments

**TRANSCODE=IGNORE | WARN**  
Specifies whether transcoding errors end SAS processing.

- Errors can also occur when transcoding the locale’s character set into the character set that is used in the SAS session.  
- Transcoding errors can occur if characters in the source data cannot be converted into the character set that is used by the selected locale.

**IGNORE**  
Prevents writing of transcoding warning messages to the SAS log. SAS processing continues and ignores any transcoding errors.

**WARN**  
Writes transcoding error messages to the SAS log, and SAS stops processing.

**Default** A value is not supplied for the TRANSCODE= option.
IWA=NO | YES
specifies whether Integrated Windows Authentication is enabled. When IWA is enabled, the SAS administrator adds IWA=YES to the configuration file of the SAS Workspace Server. When a SAS Data Quality Server procedure or function attempts to connect to a DataFlux Data Management Server, it checks DQOPTIONS. If IWA=YES, then the procedure or function loads the code that supports IWA.

NO
specifies that connections from SAS Workspace Server to DataFlux Data Management Server use the function DMSVRUSER.

YES
specifies that connections to DataFlux Data Management Server generate passwords to support IWA, without using the function DMSVRUSER. Passwords are generated once for each server connection.

Restriction
The SAS Workspace Server and DataFlux Data Management Server must be registered on the same instance of SAS Metadata Server.

**DQSETUPLOC= System Option**
Specifies the location of the root directory of the Quality Knowledge Base.

**Valid in:**
- Configuration file
- SAS invocation
- OPTIONS statement
- SAS System Options window

**Category:**
Input Control: Data Quality

**PROC OPTIONS**
GROUP= DATAQUALITY

**Syntax**

```
DQSETUPLOC=('quality-knowledge-base-root-directory')
```

**Required Argument**

*quality-knowledge-base-root-directory*
identifies the directory that is the root of the Quality Knowledge Base.
Appendix 1

Deprecated Language Elements

Dictionary

PROC DQSRVADM ................................................................. 133
PROC DQSRVSVC ............................................................... 133
DQSRVARCHJOB Function .................................................. 134
DQSRVCOPYLOG Function ................................................ 134
DQSRVDELETELOG Function ................................................. 134
DQSRVJOBSTATUS Function ................................................ 134
DQSRVKILLJOB Function .................................................... 135
DQSRVPROFJOBFILE Function ............................................. 135
DQSRVPROFJOBREP Function .............................................. 135
DQSRVUSER Function ....................................................... 135
DQSRVVER Function .......................................................... 136

Dictionary

PROC DQSRVADM

The DQSRVADM procedure creates a data set that provides the name, type, and description of all DataFlux dfPower Architect and DataFlux dfPower Profile jobs.

See Also

- PROC DMSRVADM
- *SAS Data Quality Server 9.2*

PROC DQSRVSVC

Deprecated. The DQSRVSVC procedure runs a DataFlux dfPower Architect real-time service on a DataFlux Integration Server.

See Also

- PROC DMSRVDATASVC
SAS Data Quality Server 9.2

DQSRVARCHJOB Function

Deprecated. Runs a DataFlux dfPower Architect job on a DataFlux Integration Server and returns a job identifier.

See Also

- DMSRVPROFILEJOB function
- DMSRVBATCHJOB function
- SAS Data Quality Server 9.2

DQSRVCOPYLOG Function

Deprecated. Copies a job's log file from a DataFlux Integration Server.

See Also

- DMSRVCOPYLOG function
- SAS Data Quality Server 9.2

DQSRVDELETELOG Function

Deprecated. Deletes a job's log file from a DataFlux Integration Server.

See Also

- DMSRVDELETELOG function
- SAS Data Quality Server 9.2

DQSRVJOBSTATUS Function

Deprecated. Returns the status of a job that was submitted to a DataFlux Integration Server.

See Also

- DMSRVJOBSTATUS function
- SAS Data Quality Server 9.2
**DQSRVKILLJOB Function**

Deprecated. Terminates a job that is running on a DataFlux Integration Server.

**See Also**

- DMSRVKILLJOB function
- *SAS Data Quality Server 9.2*

**DQSRVPROFJOBFILE Function**

Deprecated. Runs a file-type DataFlux dfProfile job on a DataFlux Integration Server and returns a job identifier.

**See Also**

- DMSRVPROFILEJOB function
- DMSRVBATCHJOB function
- *SAS Data Quality Server 9.2*

**DQSRVPROFJOBREP Function**

Deprecated. Runs a repository–type DataFlux dfProfile job on a DataFlux Integration Server and returns a job identifier.

**See Also**

- DMSRVPROFILEJOB function
- DMSRVBATCHJOB function
- *SAS Data Quality Server 9.2*

**DQSRVUSER Function**

Deprecated. Registers a user on a DataFlux Integration Server.

**See Also**

- DMSRVUSER function
- *SAS Data Quality Server 9.2*
DQSRVVER Function

Deprecated. Returns the version of the DataFlux Integration Server.

See Also

- DMSRVVER function
- *SAS Data Quality Server 9.2*
Recommended Reading

Here is the recommended reading list for this title:

- *SAS Language Reference: Concepts*
- *Base SAS Procedures Guide*
- *SAS Statements: Reference*
- *SAS System Options: Reference*
- *SAS Macro Language: Reference*
- SAS DataFlux documentation is available under each product’s name on the support site’s [SAS Documentation](http://support.sas.com/documentation). See also the following documents:
  - *DataFlux Data Management Studio: User’s Guide*
  - *DataFlux Data Management Server: Administrator’s Guide*
  - *DataFlux Expression Language Reference Guide*
  - Depending on your role, you might want to review the SAS Federation Server, DataFlux Authentication Server, and DataFlux Secure documentation for users or administrators.
  - See the *SAS Data Integration Studio: User’s Guide* to learn about working with DataFlux Data Management Platform and data quality transformations, especially the information about creating match codes and applying lookup standardization.

For a complete list of SAS publications, go to [sas.com/store/books](http://sas.com/store/books). If you have questions about which titles you need, please contact a SAS Representative:

**SAS Books**
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-0025
Fax: 1-919-677-4444
Email: sasbook@sas.com
Web address: [sas.com/store/books](http://sas.com/store/books)
Glossary

analysis data set
in SAS data quality, a SAS output data set that provides information about the degree of divergence in specified character values.

case definition
a part of a locale that is referenced during data cleansing to impose on character values a consistent usage of uppercase and lowercase letters.

cleanse
to improve the consistency and accuracy of data by standardizing it, reorganizing it, and eliminating redundancy.

cluster
in SAS data quality, a set of character values that have the same match code.

composite match code
a match code that consists of a concatenation of match codes from values from two or more input character variables in the same observation. A delimiter can be specified to separate the individual match codes in the concatenation.

compound match code
a match code that consists of a concatenation of match codes that are created for each token in a delimited or parsed string. Within a compound match code, individual match codes might be separated by a delimiter.

data analysis
in SAS data quality, the process of evaluating input data sets in order to determine whether data cleansing is needed.

data cleansing
the process of eliminating inaccuracies, irregularities, and discrepancies from data.

data quality
the relative value of data, which is based on the accuracy of the knowledge that can be generated using that data. High-quality data is consistent, accurate, and unambiguous, and it can be processed efficiently.

data transformation
in SAS data quality, a cleansing process that applies a scheme to a specified character variable. The scheme creates match codes internally to create clusters. All
values in each cluster are then transformed to the standardization value that is specified in the scheme for each cluster.

**delimiter**
- a character that serves as a boundary that separates the elements of a text string.

**gender definition**
- a part of a locale that is referenced during data cleansing to determine the gender of individuals based on the names of those individuals.

**guess definition**
- a part of a locale that is referenced during the selection of the locale from the locale list that is the best choice for use in the analysis or cleansing of the specified character values.

**identification definition**
- a part of a locale that is referenced during data analysis or data cleansing to determine categories for specified character values.

**locale**
- a setting that reflects the language, local conventions, and culture for a geographic region. Local conventions can include specific formatting rules for paper sizes, dates, times, and numbers, and a currency symbol for the country or region. Some examples of locale values are French_Canada, Portuguese_Brazil, and Chinese_Singapore.

**locale list**
- an ordered list of locales that is loaded into memory prior to data analysis or data cleansing. The first locale in the list is the default locale.

**match**
- a set of values that produce identical match codes or identical match code components. Identical match codes are assigned to clusters.

**match code**
- an encoded version of a character value that is created as a basis for data analysis and data cleansing. Match codes are used to cluster and compare character values.

**match definition**
- a part of a locale that is referenced during the creation of match codes. Each match definition is specific to a category of data content. In the ENUSA locale. For example, match definitions are provided for names, e-mail addresses, and street addresses, among others.

**name prefix**
- a title of respect or a professional title that precedes a first name or an initial. For example, Mr., Mrs., and Dr. are name prefixes.

**name suffix**
- a part of a name that follows the last name. For example, Jr. and Sr. are name suffixes.

**parse**
- to analyze text, such as a SAS statement, for the purpose of separating it into its constituent words, phrases, punctuation marks, values, or other types of information. The information can then be analyzed according to a definition or set of rules.
parse definition
a part of a locale that is referenced during the parsing of character values. The parse definition specifies the number and location of the delimiters that are inserted during parsing. The location of the delimiters depends on the content of the character values.

parse token
a named element that can be assigned a value during parsing. The specified parse definition provides the criteria that detect the value in the string. After the value is detected and assigned to the token, the character value can be manipulated using the name of the token.

parsed string
in SAS data quality, a text string into which has been inserted a delimiter and name at the beginning of each token in that string. The string is automatically parsed by referencing a parse definition.

Quality Knowledge Base
a collection of locales and other information that is referenced during data analysis and data cleansing. For example, to create match codes for a data set that contains street addresses in Great Britain, you would reference the ADDRESS match definition in the ENGBR locale in the Quality Knowledge Base.

scheme
a reusable collection of match codes and standardization values that is applied to input character values for the purposes of transformation or analysis.

scheme format
schemes are created and applied as files. The scheme files are created in the QKB format or the NOQKB format by the DQSCHEME procedure.

sensitivity
in SAS Data Quality, a value that specifies the amount of information in match codes. Greater sensitivity values result in match codes that contain greater amounts of information. As sensitivity values increase, character values must be increasingly similar to generate the same match codes.

standardization definition
a part of a locale that is referenced during data cleansing to impose a specified format on character values.

standardize
to eliminate unnecessary variation in data in order to maximize the consistency and accuracy of the data.

token
in SAS data quality, a named word or phrase in a parsed or delimited string that can be individually analyzed and cleansed.

transformation
in data integration, an operation that extracts data, transforms data, or loads data into data stores.

transformation value
in SAS data quality, the most frequently occurring value in a cluster. In data cleansing, this value is propagated to all of the values in the cluster.
Index

**Special Characters**
- ! (exclamation point) as delimiter 39
- %DQPUTLOC autocall macro 6, 70
- %DQUNLOAD autocall macro 72

**A**
- analysis data sets 61
  - creating 58, 62
- ANALYSIS= option
  - CREATE statement (DQSCHEME) 58
  - apply mode 62
- APPLY Statement
  - DQSCHEME procedure 55
- authentication 27, 33, 89

**B**
- blank spaces, removing 28
  - blank values 38
- BLOCKSIZE= option
  - PROC DMSRVDATASVC statement 26

**C**
- CALL routines 74
  - DQSCHEMAAPPLY 120
  - scheme CALL routines 79
- CAS
  - access data in 7
  - case definitions 17, 91
  - case functions 76
  - case of character values 91
  - category names
    - from character values 99
  - character values
    - after standardization 125
    - case and standardization definitions 17
    - case of 91
    - category names from 99
    - gender analysis, locale guess, and identification definitions 18
  - inserting tokens into extraction values 95
  - inserting tokens into parsed values 114
  - locale names from 100
  - match codes from 104
  - match codes from parsed values 106
  - parsed 109
  - pattern analysis definitions 19
  - pattern analysis from input values 116
  - tokens from 126
  - tokens from extraction values 94
  - tokens from parsed values 113
  - transporting 9
  - updated extraction values 95
  - updated parsed values 114
  - cleaning up jobs and logs 24
  - cleansing data
    - real-time 25
  - Cloud Analytic Server
    - access data in 7
    - cluster numbers 37
      - assigning 41
      - creating 43
    - CLUSTER_BLANKS option
      - PROC DQMATCH statement 38
    - CLUSTER= option
      - PROC DQMATCH statement 38
    - clustering with exact criteria
      - DQMATCH procedure, CRITERIA statement 12
    - clusters 11
      - householding 12
      - minimal sensitivity 46
      - mixed sensitivity 44
      - with exact criteria 12
      - with multiple CRITERIA statements 49
    - CLUSTERS_ONLY option
      - PROC DQMATCH statement 39
    - composite match codes 11, 43
    - CONDITION= option
      - CRITERIA statement (DQMATCH) 40
      - configuration
      - SAS sessions for data quality 6
    - CONVERT Statement
Index

DQSCHEME procedure 56
CONVERT statement, DQSCHEME procedure 57
converting schemes 57
count of locale definitions 102
CREATE statement
DQSCHEME procedure 58
CREATE Statement
DQSCHEME procedure 57
CRITERIA statement
DQMATCH procedure 40
DQMATCH Procedure 40
CRITERIA statement, DQMATCH procedure
clustering with multiple 49

D
data cleansing
real-time 25
specifying definitions in programs 6
data quality
SAS session parameters for programs 131
data sets
analysis 58, 61, 62
creating 58
input 28, 34
job status 21, 22, 24
output 29, 34
scheme 58
data values, similar
transforming 53
DATA= option
PROC DMSRVDATASVC statement 27
PROC DQMATCH statement 39
PROC DQSCHEME statement 55
DataFlux Data Management
running services 34
DataFlux Data Management Profile
running profile jobs 87
DataFlux Data Management Server
authenticating users 27, 33, 89
cleaning up jobs and logs 24
copying log files 81
deleting log files 83
functions 76
generating Profile jobs 87
host machine 27, 32
host of 22
input and output columns 28, 33
job information 21
job status 84
jobs and services 7
passwords 8
port number 22, 27, 33
processing services 31
running Data Management Studio jobs 80
running jobs and services 7
running real-time services 25
security 23
service identification 28, 33
terminating jobs 86
version of 90
DataFlux Data Management Studio
creating jobs and services 7
job identifiers 80
running jobs 80
running real-time services 25
running services 29, 30
DataFlux Profile jobs 87
date standardization
in EN locale 18
default length of parsed input 111
definitions
available in locales 6
case 91
case and standardization 17
date standardization in EN locale 18
displaying information about locale 6
extraction 16
gender 97
gender analysis, locale guess, and identification 18
match definitions 17
parse 15
pattern analysis 19
revealing or hiding non-surfaced 107
specifying in data cleansing programs 6
delimiter
exclamation point as 39
DELIMITER option
PROC DQMATCH statement 39
DELIMSTR= option
CRITERIA statement (DQMATCH) 41
DMSRVADM procedure 21
cleaning up jobs and logs 24
examples 24
job status data sets 22, 24
PROC DMSRVADM statement 22
security and 23
syntax 22
DMSRVADM Procedure
syntax 21
DMSRVBATCHJOB function 80
DMSRVCOPYLOG function 81
DMSRVDATASVC procedure
examples 29, 30
input and output data sets 28
syntax 26
timeout 28
DMSRVDATASVC Procedure 25
syntax 25
DMSRVDELETELOG function 83
DMSRVJOBSTATUS function 84
DMSRVKILLJOB function 86
DMSRVPROCESSSVC procedure examples 34
input and output data sets 34
DMSRVPROCESSSVC Procedure 31
syntax 31
DMSRVPROFILEJOB function 87
DMSRVUSER function 89
DMSRVVER function 90
DQCASE function 91
DQEXTINFOGET function 92
DQEXTRACT function 93
DQEXTTOKENGET function 94
DQEXTTOKENPUT function 95
DQGENDER function 96
DQGENDERINFOGET function 97
DQGENDERPARSED function 98
DQIDENTIFY function 99
DQLocale= system option 130
DQLOCALEGUESS function 100
DQLOCALEINFOGET function 102
DQLOCALEINFOLIST function 102
DQLOCLST Procedure 35
DQMATCH function 104
DQMATCH procedure 37
clustering with minimal sensitivity 46
clustering with mixed sensitivities 44
clustering with multiple CRITERIA statements 49
CRITERIA statement 40, 49 examples 43
generating composite match codes 43
generating multiple simple match codes 50
householding 12
match codes for parsed values 47
matching values with mixed sensitivity levels 44
PROC DQMATCH statement 38 syntax 38
DQMATCH Procedure syntax 38
DQMATCH statement 38
DQMATCHCHINFOGET function 105
DQMATCHCHPARSED function 106
DQOPTIONS= system option 131 configuring SAS sessions 6
DQOPTSURFACE function 107
DQPARSE CALL routine 109
DQPARSEINFOGET function 110
DQPARSEINPUTLEN function 111
DQPARSERESLIMIT function 112
DQPARSECORDEPTH function 113
DQPARSETOKENGET function 113
DQPARSETOKENPUT function 114
DQPATTERN function 116
DQSCHEME procedure 53
APPLY statement 55
applying schemes 66
CONVERT statement 57
CREATE statement 58
creating analysis data sets 62
creating QKB schemes 65
creating schemes 64
date standardization in 18
encoded passwords 8
EXACT option CRITERIA statement (DQMATCH) 41
exclamation point as delimiter 39
extraction character values inserting tokens into 95
tokens from 94
updated 95
extraction definition functions 78
extraction definitions extract 16
token names in 92
extraction input 16
extraction values updated 95

E
EN locale date standardization in 18
encoded passwords 8
EXACT option
CRITERIA statement (DQMATCH) 41
exclamation point as delimiter 39
extraction character values inserting tokens into 95
tokens from 94
updated 95
extraction definition functions 78
extraction definitions extract 16
token names in 92
extraction input 16
extraction values updated 95

F
functions 74
case 76
DataFlux Data Management Server 76
extraction 78
gender analysis, locale guessing, and identification 77
listed alphabetically 74
listed by category 76
matching 77
parsing 77
pattern analysis 78
reporting 78
scheme 79
standardization 79

G
gender analysis definitions 18
gender analysis functions 77
gender definitions
  parse definitions associated with 97
gender determination
  from name of an individual 96
  from parsed name 98
generating jobs
  Profile jobs 87
global parse
  global 16

H
hiding non-surfaced definitions 107
host
  for DataFlux Data Management Server 22
host machine
  DataFlux Data Management Server 27, 32
HOST= option
  PROC DMSRVADM statement 22
  PROC DMSRVDATASVC statement 27
  PROC DMSRVPROCESSSSVC
    statement 32

I
identification definitions 18
identification functions 77
IGNORE_CASE option
  APPLY statement (DQSCHEME) 56
  CREATE statement (DQSCHEME) 59
IN= option
  CONVERT statement (DQSCHEME) 57
INCLUDE_ALL option
  CREATE statement (DQSCHEME) 58
input
  default length of parsed 111
  extraction values 16
  parsed values 15
input character values
  pattern analysis from 116
input columns, input and output 28, 33
input data sets
  DMSRVDATASVC procedure 28
  DMSRVPROCESSSSVC procedure 34
installation 6

J
job identifiers
  DataFlux Data Management Studio jobs 80
  Profile jobs 87
job status data sets 21, 22
generating 24
  location of 22
jobs
  cleaning up 24
  copying log files 81
  creating 7
  deleting log files 83
  generating Profile jobs 87
  information about 21
  running 7
  running DataFlux Data Management
    Studio jobs 80
  status of 84
  terminating 86

K
killing jobs 86
  terminating 7

L
length
  default length of parsed input 111
locale definitions
  case and standardization 17
  count of 102
  date standardization in EN locale 18
  gender analysis, locale guess, and
    identification 18
  global parse 16
  locale 15
  match 17
  name of 102
  parse 15
  pattern analysis 19
  locale guess definitions 18
  locale guessing functions 77
  locale names
    from character values 100
  LOCALE= option
    APPLY statement (DQSCHEME) 55
    CREATE statement (DQSCHEME) 58
PROC DQMATCH statement 39
locales
definitions available in 6
displaying information about 6
displaying information in SAS log 70
getting information about 102
loading and unloading 9
ordered list of 130
unloading to use free memory 72
updating 9
log files
cleaning up 24
copying 81
deleting 83
displaying locale information 70
for jobs and services 7
lookup method 56, 59, 62

M
code
match codes 37
Composite 11, 43
creating 10, 40
creating for parsed values 47
from character values 104
from parsed character values 106
generating multiple simple codes 50
length of 11
match definitions and 17
sensitivity 12
simple 11, 50
truncation of 11
match definitions
parse definitions associated with 105
MATCH-DEFINITION= option
APPLY statement (DQSCHEME) 56
MATCHCODE= option
CRITERIA statement (DQMATCH) 42
PROC DQMATCH statement 40
MATCHDEF= option
CREATE statement (DQSCHEME) 58
CRITERIA statement (DQMATCH) 41
output columns
matching functions 77
matching values
default sensitivity 43
minimal sensitivity 46
mixed sensitivity 44
memory
unloading locales from 72
meta options 61
MISSINGVARSOK option
PROC DMSRVDATASVC statement 27
mode of scheme application 56, 58
MODE=ELEMENT option
APPLY statement (DQSCHEME) 56
CREATE statement (DQSCHEME) 58

N
named tokens 15, 16, 92, 110
names
gender determination and 96, 98
locale names from character values 100
of locale definitions 102
of parse definitions 97, 105
parsed 98
NO CLUSTER_BLANKS option
PROC DQMATCH statement 38
NODELIMITER option
PROC DQMATCH statement 39
non-surfaced definitions
revealing or hiding 107
NOPRINT option
PROC DMSRVDATASVC statement 27
PROC DMSRVPROCESSSVC statement 32

O
OUT= option
CONVERT statement (DQSCHEME) 57
PROC DMSRVADM statement 22
PROC DMSRVDATASVC statement 27
PROC DMSRVPROCESSSVC statement 33
PROC DQMATCH statement 40
PROC DQSCHEME statement 55
output columns 28, 33
output data sets
DMSRVDATASVC procedure 29
DMSRVPROCESSSVC procedure 34

P
PARMLIST= option
PROC DMSRVDATASVC statement 27
PROC DMSRVPROCESSSVC statement 32
parse definitions 15, 16
associated with gender definitions 97
associated with match definitions 105
name of 97, 105
token names in 110
parsed character values 109
inserting tokens into 114
match codes from 109
tokens from 113
updated 114
parsed input 15
default length of 111
parsed names
gender determination from 98
parsed values
match codes for 47
updated 114
parsing
resource limit during 112
parsing functions 77
parsing scores
searching for 113
PASSWORD= option
PROC DMSRVDATASVC statement 27
PROC DMSRVPERSVCstatement 33
passwords
for DataFlux Data Management Server 8
pattern analysis
from input character values 116
pattern analysis definitions 19
pattern analysis functions 78
PHRASE option
APPLY statement (DQSCHEME) 56
CREATE statement (DQSCHEME) 58
port number 22, 27, 33
PORT= option
PROC DMSRADM statement 22
PROC DMSRVDATASVC statement 27
PROC DMSRVPERSVC
statement 33
PROC DMSRADM statement 22
DMSRADM Procedure 21
PROC DMSRVDATASVC statement 26
DMSRVDATASVC Procedure 26
PROC DMSRVPERSVC
DMSRVPERSVC Procedure 32
PROC DQLOCST
DQLOCST Procedure 35
PROC DQMATCH statement 38
procedure 53
process
Data Management service 31
service 31
QKB file format
converting SAS schemes to 57
creating QKB schemes 65
version of SAS Data Quality engine 127
QKBTOQAS option
CONVERT statement (DQSCHEME) 57
Quality Knowledge Base (QKB)
download latest 1
location of root directory 132
system access 5
version of currently loaded 128
R
real-time data cleansing 25
reporting functions 78
resource limit
during parsing 112
revealing non-surfaced definitions 107
root directory
specifying location of 132
running jobs 7
DataFlux Data Management Studio 80
running services 7
Data Management Server 31
real-time 25
S
SAS Data Quality Server
capabilities 1
concepts 5
installing 6
integration components 2
Quality Knowledge Base 5
software license 2
updating 6
SAS format schemes
convert scheme file from QKB to SAS format 57
SAS log
displaying locale information 70
SAS Macro resources
SAS sessions
configuring for data quality 6
parameters for data quality programs 131
SASTOQKB option
CONVERT statement (DQSCHEME) 57
scheme CALL routines 79
scheme functions 79
SCHEME_LOOKUP= EXACT option
APPLY statement (DQSCHEME) 56
CREATE statement (DQSCHEME) 59
SCHEME= option
APPLY statement (DQSCHEME) 56
CREATE statement (DQSCHEME) 59
schemes
standardization 125
  date standardization in EN locale 18
standardization definitions 17
standardization functions 79
system options 129

T
  terminating jobs 86
    logs for 7
TIMEOUT= option
    PROC DMSRVDATASVC statement 28
    PROC DMSRVPROCESSSSVC statement 33
token names 15, 16, 92, 110
tokens
  from character values 126
  from extraction character values 94
  from parsed character values 113
  inserting into extraction character values 95
  inserting into parsed character values 114
  updated extraction character values 95
  updated parsed character values 114
transforming
  applying schemes to transform values of a single variable 55
TRIM option
  PROC DMSRVDATASVC statement 28
truncated match codes 11

U
  unloading locales 72
  updated extraction values 95
  updated parsed values 114
  updated tokens 95, 114
  updating
    SAS Data Quality Server 6
USE_MATCHDEF option
  APPLY statement (DQSHEME) 56
  CREATE statement (DQSHEME) 59
user authentication 27, 33, 89
USERID= option
  PROC DMSRVDATASVC statement 28
  PROC DMSRVPROCESSSSVC statement 33

V
  VAR= option
    APPLY statement (DQSHEME) 56
CREATE statement (DQSCHEME) 60
CRITERIA statement (DQMATCH) 41
variables
applying schemes to transform values of a single variable 55
version
currently loaded QKB 128
DataFlux Data Management Server 90
QKB scheme 127